IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 19, 2018, accepted December 2, 2018, date of publication December 13, 2018,
date of current version January 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2886639

Aligning Business Processes With the Services Layer Using a Semantic Approach

ENCARNA SOSA-SÁNCHEZ[1], (Fellow, IEEE), PEDRO J. CLEMENTE[1], ÁLVARO E. PRIETO[1], AND CARMEN ORTIZ-CARABALLO[2]

[1]Department of Computer Science, University of Extremadura, 10003 Cáceres, Spain
[2]Mathematics Department, Campus d'Igualada, University of Lleida, 08700 Igualada, Spain

Corresponding author: Encarna Sosa-Sánchez (esosa@unex.es)

This work was funded by (i) Ministerio de Economía e Innovación (Spain) TIN2015-69957-R (MINECO/ERDF, EU) project, (ii) POCTEP 4IE project (0045-4IE-4-P), and (iii) Consejería de Economía e Infraestructuras/Junta de Extremadura (Spain) - European Regional Development Fund (ERDF)-GR15098 project and IB16055 project.

ABSTRACT Organizations require their business processes goals and the underlying information technology (IT) to be in synchronization with each other, but the continual changes in business processes makes this difficult. To accomplish this synchronization, there needs to be an alignment between the business processes and the IT. Business processes are currently defined using such well-known notations as BPMN, and the IT is made available by different services. Hence, the alignment process can be defined as one between the organization's BPMNs and the services provided by its IT. In practice, however, this process is a complex task which is carried out by hand and hence is error prone. The present communication analyzes the conditions, relations, and incompatibilities between BPMNs and the service descriptions. The incompatibilities are formalized mathematically in order to facilitate their identification and resolution. Then, an alignment process is defined taking into account these incompatibilities and their solutions. The wrapper code needed to resolve each incompatibility identified during the alignment process is generated automatically. Finally, a case study is presented to validate and illustrate the use of the proposed alignment process. The results provided by the semiautomatic alignment process were similar to those obtained manually by a group of experts.

INDEX TERMS Alignment process support, business process alignment, service-oriented architecture, semantic algorithms, service incompatibility resolution.

I. INTRODUCTION

Business and information technology (IT) Alignment refers to the optimized alignment of business processes (BPs) and their respective technological services provided by IT [1], [2]. Firms and organizations are becoming increasingly aware of the need to keep their BPs aligned with their technological infrastructure. Organizations that do so have a greater tolerance to the changes that are common in the business and institutional world. These changes are due to many factors [1], [3]. Examples are market changes, changes in regulations and standards, changes in firm policy, etc. In this sense, it is critical to have mechanisms that facilitate the adaptation of both the BPs and the information systems that support them [4]. So any change in the BP descriptions should be aligned with the firm's information technology (IT). This is a complex and demanding task for IT professionals who have to provide services to business organizations when those organizations are moving rapidly towards new goals and objectives [5], [6]. Ultimately, a firm's performance is strongly correlated with the maturity of its alignment processes [1].

Large firms may have defined hundreds or thousands of BPs that need to be managed. Hence, intelligent software is required to facilitate the execution of such common processes as BP storage, BP search, or BP version management, inter alia [7]. Besides this business process management, an alignment process should be developed between BPs and the deployment artifacts (such as services) [8]. Although an alignment process could success in both ways, that is, from IT to BP and from BP to IT, usually BP to IT is considered the common approach because IT infrastructure should answer to BP requirements.

Business Process Management (BPM) is a systematic approach to managing organizational processes, making them more effective, efficient, and reliable when faced with

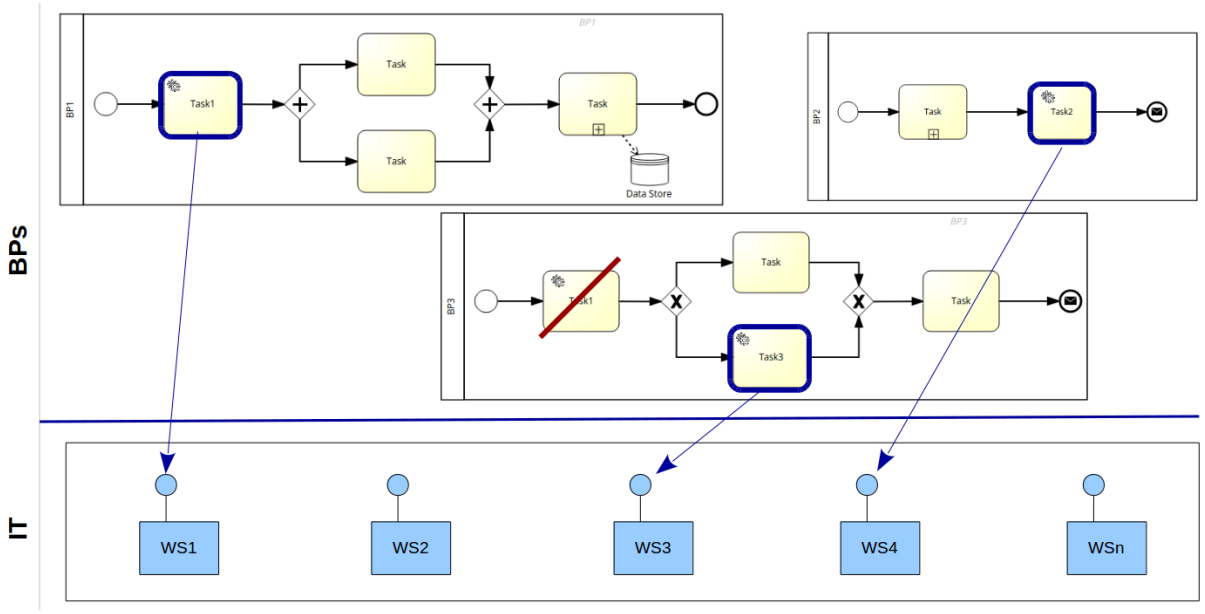

FIGURE 1. Alignment between Business Processes (BP) and Infrastructure Technology (IT).

changes in the business environment [1]. According to [9], these processes constitute an interconnected set working in a sensible flow to ensure that the main business objectives are met. They may be defined using BPMN [10], a notation commonly used in BPM. It allows the interactions among tasks, data structures definition, and exception management to be described. Note that BPs should be defined independently of the specific services which finally implement their tasks [11]. This allows different teams to focus on the essential aspects of the BPs defined in the firm without having to concern themselves with possible implementations. Furthermore, the BPs can change over time, and their independence of the underlying technology is essential in order to facilitate the firm's adaptation to new challenges and goals [1]. Service-oriented architectures (SOAs) have become widely used over the last few years in order to improve the development, deployment, and management paradigm of information systems that need to be aligned with business processes, and where the underlying services are adaptable and reusable [12]. Specifically, a SOA is based on a set of design principles such as modularity and loose coupling, implementation independence, open standards, service descriptions, interoperability, platform independence, and service contract [13]. A SOA must decouple business applications from technical services, and make the enterprise to be independent of any specific technical implementation or infrastructure [11]. It is therefore a paradigm that facilitates the deployment of systems following the organization's BPs [14], [15].

Thus, the combination of BPM and SOA is now being seen as one of the most advanced approaches to integrating business and IT layers [12]. In this sense, whereas BPM allows managers to specify BP rules, SOA acts as the architectural model with which such rules are transformed into systems. In terms of implementation, Web service technology is seen as the facto standard for SOA-based systems [13]. Today, services are independently developed and deployed in a variety of technologies. Particularly important from a business perspective is that they can be composed freely. Moreover, a set of tools can be used to orchestrate services, and these orchestrations may correspond to the implementation of a BP [15]. Note that, in the current global context of software development, services could be produced and consumed as software-as-a-service (SaaS) [16]. Hence, it is interesting to have a hybrid software model which defines orchestrated services by means of BPM approaches that also include third party services (SaaS).

As explained above, there is a need to align BPs with the underlying services. This implies a transformation from logical to physical business models [17] that should be carried out by means of an alignment process. It has to be noted that the information system implementation driven by the BPs is the highest level of a SOA maturity pyramid [18]. In this sense, in order to obtain an automatic alignment process between BPs and the underlying services, the BPs can not only be represented graphically for documentation purposes. On the contrary, it is necessary to define them using a standard notation, following the operational BP specification guidelines [19] which includes descriptions of the interactions among tasks, data structures definition, input and output operations parameters and exception management. Figure 1 shows this alignment process from BP to IT. From then on, the software architect should search for services which allow each task of the BP to be implemented.

However, the alignment processes are carried out by hand, a procedure that is both tedious and error prone. There is also the additional difficulty of identifying incompatibilities

between the business processes tasks (BPTs) definitions and the analyzed services. Also, on the one hand, the number of BPs and the number of services stored in a repository could be very large, so that the manual task of service identification would be very difficult. And, on the other hand, the complexity of defining by hand the service orchestration which must implement the BPs should be considered, as also should that of implementing service wrappers to facilitate the alignment. For these reasons, there is a need for methods, languages, and tools which facilitate the process of alignment between BPTs and the underlying services.

This alignment process identifies the most suitable services to implement each BPT. It should also be emphasized that this selection will be done by means of semantic algorithms. Furthermore, a set of incompatibilities between BPTs and services will be analyzed and identified during the alignment process. It will then be possible to automatically generate a set of possible adaptations to solve some of these incompatibilities. Thus, this work extends [20] including an extended alignment process description, a mathematical formulation of the incompatibilities managed and the wrappers code definition needed to adapt the underlying web services.

The main contributions of the present communication are:

- A definition of an alignment process which uses semantic algorithms in order to identify the most suitable service for each BPT, i.e., to develop the alignment process.
- A mathematical formulation to analyze, identify and solve a set of alignment incompatibilities between BPTs and services. These incompatibilities are resolved by the alignment process.
- An alignment process between the BPTs and the selected services, including, whenever necessary, the description of the adaptations needed to solve the rectifiable incompatibilities.

Furthermore, an experimental case study that was carried out to validate the process is also presented, comparing the results provided by our algorithm with those provided by a group of experts. The following set of research questions was defined in accordance with guidelines for designing case studies [21]:

- Main Research question RQ_0: Can an automatic alignment process between BPTs and the underlying services obtain results similar to those of the alignment carried out manually by an expert?
- Research question RQ_1: Is the dictionary used during the alignment process valid with respect to the BP domain?
- Research question RQ_2: Is the effort required to identify a potential alignment between a task and a service greater when the service parameter names, their order, or their data types are substantially different from those defined in the task?
- Research question RQ_3: Is the effort required to identify a potential alignment between a task and a service greater when they have different numbers of parameters?

The paper is structured as follows: Section 2 identifies and compares related works; Section 3 presents the analysis about different incompatibilities types that are taken into account; in Section 4, the alignment process and incompatibility issues are formulated mathematically; Section 5 presents the results of the method and compares them with the experts' results; and, finally, Section 6 presents the main conclusions.

II. RELATED STUDIES

Complete automation of BP alignment is a complex process. However, there are several studies that have dealt with the search for and identification of services and their incompatibility issues. Hence, we studied some of these service discovery approaches in order to design an automatic alignment process of BPs with the underlying services. Specifically, two main aspects were taken into account in this review. Firstly, what are the main characteristics and limitations of a service discovery process based on a semantic algorithm to implement an alignment process? And secondly, what are the mechanisms used to detect incompatibilities between the services used and provided?

A. SERVICE DISCOVERY APPROACHES

This subsection describes various approaches related to the service discovery process whose main objective is to offer a suitable Web service candidate resulting from a specific search. Hence, these studies mainly include service discovery approaches based on WSDL (Web Services Description Language) Web services, semantic Web services, and a proposal for a semantic SOA. I.e., from the semantic information of each Web service, one obtains a semantic SOA in which the semantic information describes the SOA itself.

In [22], a service discovery process is defined taking into account the context of the application of the BPs, and including several non-functional requirements such as quality of service. This service discovery process involves a semantic algorithm whose inputs are: (i) services defined in UDDI (Universal Description, Discovery, and Integration) or federated services, (ii) BPs defined by the UBL (Universal Business Language) ontology, and (iii) a search string related to the service being searched for. The process returns a list of candidate services that perform semantic matching with the UBL and QoS (Quality of Service) match functions. However, the authors provide no information about how the semantic matching is carried out. One has to note that non-functional service information is not always available, and these methods (non-functional properties) can only serve as a complement to service discovery.

Narock *et al.* [23] define a service provenance ontology in order to increase the efficiency of service discovery. This ontology integrates W3C Provenance Ontology (PROV-O) [24], the V7 Model [25], and additional Web service related concepts. Thus, for each service they require such information as the methods used, the applications, actions, and settings invoked, and the assumptions and hypotheses involved. As a consequence, during the discovery, the service provenance can tell users what a service can do. Although this approach is interesting and can improve the service discovery results, the information available in the service repository

is usually limited to syntactic information. In the present work, we align BPs with underlying services which only have syntactic information available.

Other approaches to implementing service discovery include: (i) a simple keyword-based and category-based search on UDDI; (ii) identifying with Woogle [26] in WSDL artifacts that recommend similar services; and (iii) the specialized framework proposed by Hatzi *et al*. [27] to retrieve services in both WSDL and OWL-S (Web Ontology Language for Services) standards by extending and adapting the TF-IDF (Term Frequency–Inverse Document Frequency) model. There are also approaches based on semantic languages: OWLS-MX (hybrid matchmaker for OWL-S services) [28]. In [29], SAWSDL (Semantic Annotations for WSDL) semantic annotations are used to automatically find service compositions. However, these approaches have the drawback that, since semantic Web services are not widely used, many Web service repositories do not consider them.

Li *et al.* [30] address automating the process of analyzing and ranking services to retrieve the services most closely related to the query. They analyze the features describing the functional attributes of Web services, and propose a probabilistic framework and a Web service retrieval model. Since the work does not take into account specific parameters, data types, or order, it is unsuitable for the alignment of BPTs with Web services, although it could be used to recommend Web services on the basis of a user query.

The SWSD framework [31] proposes a keyword-based discovery process for Web service searches. The services are described using semantically enriched annotations. It makes intensive use of natural language processing techniques and a WordNet-based [32] similarity measure to match keywords (using ontology-based semantic matching algorithms). However, ontologies are not widely used, and the services defined do not normally include semantic annotations.

Finally, El-Gayar and Deokar [33] take a different perspective, focusing on a semantic SOA. They present the design and implementation of a Distributed Model Management System (DMMS) which could be used to manage an SOA. They used SAWSDL and SMML specifications to annotate Web services which are managed as models. For this, a model management service is defined which includes various modules such as an ontology reasoning service, model discovery services, and a model composer service, among others. The model discovery services leverage semantic and syntactic metamodel information to identify models relevant to a particular problem, and provide support for an iterative search process allowing the user to refine their search criteria based on the results being returned. Although the proposal takes advantage of semantic Web services, no alignment process from BPs to the underlying services is defined. In this sense, one notes that it could be interesting to carry out this alignment process over a semantic SOA framework like this. However, our present focus is on offering an alignment process between BPs and syntactic Web services.

Table 1 summarizes the service discovery related studies which we have reviewed. As one observes in this table, it can be said, firstly, that the different service discovery processes are closely related to the way in which they are described (WSDL, UDDI, OWS, SAWSDL). Secondly, semantic Web services and non-functional properties are not usually available in Web service repositories. Consequently, neither does our proposal deal with these issues. Lastly, there are no approaches addressing the automatic alignment of the business processes tasks from a BPMN diagram with the underlying services.

B. WEB SERVICE INCOMPATIBILITIES

One feature of our proposal is that it addresses some incompatibilities that may occur between tasks and services. These incompatibilities could cause the rejection of services that would, with minor modifications, potentially be aligned. Since we found no studies dealing with the compatibility between BPTs and Web services, we reviewed those that dealt with the compatibility between two Web services. Our motivation was to examine whether our problem could be treated in a similar way.

In this area of research, there are basically two points of view on considering whether two Web services might be compatible or not: dynamic compatibility and static compatibility [34], [35].

On the one hand, dynamic compatibility considers the behavioral features of Web services, basically by means of describing and examining the possible sequences of messages each of them can send or receive. This compatibility is the subject of some important studies such as [34] and [36]–[38]. They all would have to be taken into account if stateful Web services were used. But they are not applicable to our approach since our focus is on stateless Web services, i.e., Web services in which requests are independent of each other.

On the other hand, static compatibility considers the structure of the messages together with their input and output parameters and their data types. This is the class of incompatibility that our approach will attempt to solve for some cases. The only work that we found which specifically focused on this problem is that of [39]. Those authors deal with structural and value incompatibilities. A structural incompatibility arises when there exists a mismatch in the structure of the message as sent by the sender and as expected by the receiver. A value incompatibility arises when there exist unexpected filled-in values. Considering applications written to a given source service S, a non-native target service T may differ from S in five specific ways: ''missing method m'' (target added m or source removed m), ''extra field f on inpu'' (target added f or source removed f), ''missing field f on input/outpu'' (source added f or target removed f), ''facet mismatch for field f on input/output'', and ''cardinality mismatch for field f on input/output''. The authors provide a tool that allows to determine whether these incompatibilities are relevant or irrelevant to a specific case, and a GUI

TABLE 1. Comparison of studies related to service discovery.

	WSDL/UDDI	Semantic Web Services	Semantic Algorithm	Business Process Alignment	Non-functional Properties
Dorm et al. [8]	X (proposal)	-	X (proposal)	-	X (proposal)
Souza et al. [22]	X	-	X	X (business process ontology)	X
Narock et al. [23]	X	X (service provenance ontology)	X	-	-
Woogle [26]	X	-	-	-	-
Ourania Hatzi et al. [27]	X	X (OWL-S)	-	-	-
OWLS-MX [28]	-	X (OWL-S)	-	-	-
SAWSDL [29]	-	X (SAWSDL)	-	-	-
Li et al. [30]	X	-	X (probabilistic)	-	-
SWSD [31]	-	X (OWL-S)	X	-	X
El-Gayar O. and Deokar A. [33]	-	X (SAWSDL and SMML)	X	-	-
Alignment process proposed in the present paper	X	-	X	X (BPMN)	-

tool to resolve incompatibilities by generating middleware components that enable interoperation. Their solution does not solve our problem of aligning BPTs and services. This is due to our goal being to discover what BPTs of our BPMN models can potentially be aligned with Web services without any kind of middleware between them. However, it is interesting to examine more closely their five types of incompatibility to see which of them might really be involved in our problem. Thus, ''missing method m'' is irrelevant to our problem because we assume that a service only provides one operation that is useful for the clients. However, ''extra field f on input'', translated to our problem, can arise when a BPT of the BPMN model has fewer input parameters than the Web service. Likewise, ''missing field f on input/output'' can arise in our problem when a BPT has more input parameters or more output parameters than the Web service. On the contrary, ''facet mismatch for field f on input/output'' is an interesting incompatibility, but not relevant to our problem because dealing with a different value space is beyond the scope of our approach. Finally, ''cardinality mismatch for field f on input/output'' is not considered because we assume that the analyzable Web services have a precise definition of the number of input/output parameters in their WSDL.

III. ANALYZING TYPES OF INCOMPATIBILITIES

Different types of incompatibilities may appear between the tasks included in the BPs and the identified services during the alignment process. This section presents the analysis of these possible incompatibilities together with the case study that we have used to help us to identify and visualize them.

Concretely, the case study that we have used is about diverse business processes related to the university context. It includes aspects such as students, teachers, library, books, loans, subjects, degrees, and salaries. A set of BPs was defined for this case. Examples are BPs to enroll students, to consult marks and grades, for book loans, to consult a teacher's salary, etc.[1]

The Web services which form the university service layer are defined using the SOAP Web Services style. Consequently, their descriptions are defined using WSDL. Examples of the services defined for this case (all of them being CRUD operations) are: manage books, manage book loans, manage degrees, manage personal information about

[1]Additional information about the BP definition (BPMN) can be found in https://sites.google.com/site/migrasoa/uex-case-study/uex-bpmn.

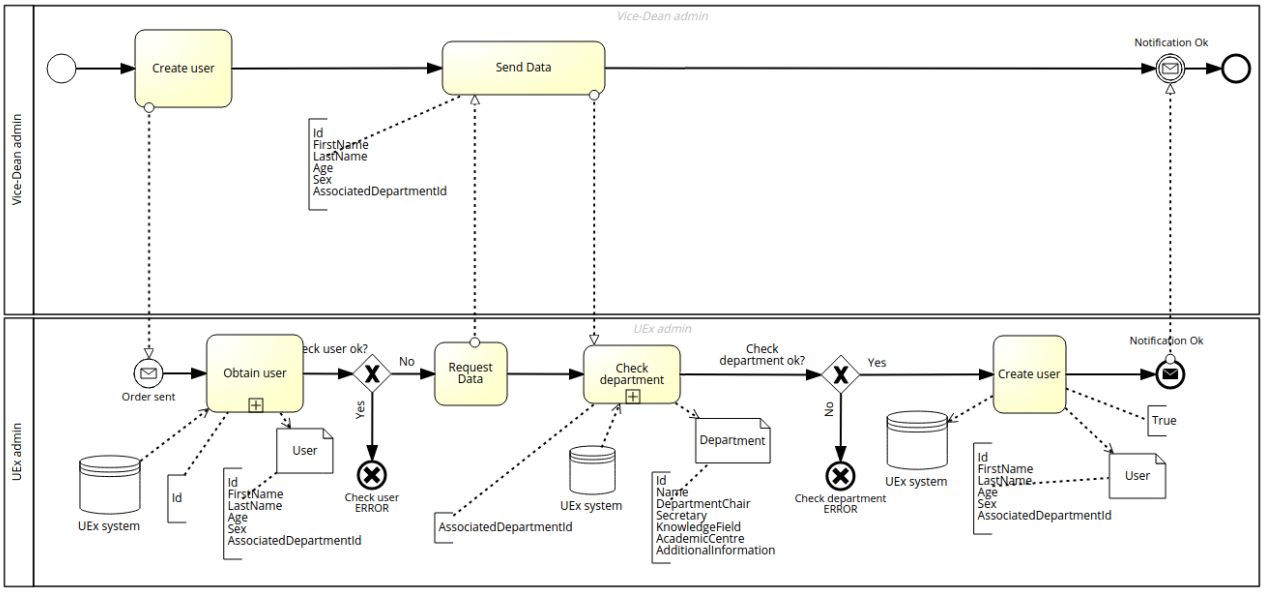

FIGURE 2. ***CreateUser*** **business process modeled using BPMN.**

teachers, students, administrative staff, etc. These services are defined from the university's legacy application.

The design of these BPs followed the guides to describe operational BPs proposed in [19] and in works such as the one by Kaindl *et al*. [40], where they explain the semantic specification of tasks in the context of business processes. I.e., these BPs include the data object details which need to be processed in each task. For example, a BPT definition includes the definitions of the input and output parameters. Note that the BP designers were independent, and were focused on attending to the organization's needs. For this reason, our approach proposes some basic writing-task rules for uniformity in style in modeling business processes. These writing-task rules mainly define how to write the expressions describing tasks, parameters, and gateways. They are summarized in Table 2. These kinds of style rules are usually established by software factories in which many people collaborate in the development of business processes. Figure 2 shows a BPMN example, specifically, the *CreateUser* Business Process.

Initially, 80 tasks (contained in the BPs) and 65 services have been identified for our case study. Once the BPs have been defined, the next step is to align the tasks they contain with the Web services that might be compatible. It should be noted that the behavior of the business processes is obtained by aligning the tasks individually. Thus, our approach proposes an alignment process that analyses the compatibility between tasks and services in three stages. In the first stage, whether the method names (both the task label and the service operation) are semantically compatible is analyzed. In the second stage, the compatibility of a pair of parameters is studied. And in the third stage, the compatibility of the whole *signature* is examined. The whole task or service name together with its parameters is called *signature*.

TABLE 2. **Basic writing-task rules used to model business processes.**

BP element	Notation
Task	Infinitive verb(*) + [-all] + object e.g. Create Conference
Notification tasks	Notification + object + OK/Error + [to whom] e.g. Notification Conference OK to admin
Check tasks	Check + object + infinitive verb(*) + [-all] e.g. Check Paper Copy
Task parameters	Parameter names separated by commas e.g. ID, location, name
Gateway labels (*) phrasal verbs are written with a dash	Infinitive verb(*) + [-all] + object + Ok +'?' e.g. Create Conference Ok?

In order to better explain the different possibilities that may arise happen in each stage, we shall consider different versions of a task denominated *Obtain User* and a service denominated *getUser*. Their complete signature is:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

A. COMPATIBILITY OF METHOD NAMES

Firstly, we will consider that the method names of a task and of a service are semantically compatible if, on the one hand,

the action verbs of the methods are the same or are synonyms, and, on the other, the object of the methods are the same or are synonyms.

Following with the example, if the *Obtain User* task and the *getUser* service have to be aligned, they can be considered to be semantically compatible based on their method names because, on the one hand, the action verbs of the methods, *obtain* and *get*, are synonyms, and, on the other, the object of both method names, *User*, is the same.

B. COMPATIBILITY OF A PAIR OF PARAMETERS

If the method names are semantically compatible, the next step is to try to align each input parameter of the task with an input parameter of the service, and each output parameter of the task with an output parameter of the service. This step is trivial when the parameters have the same name, the same data type, and the same order. We call this case Basic Compatibility. However, normally this is not true, so that it is necessary to resolve the incompatibility. An example of Basic Compatibility is the following:

- **Basic Compatibility.** The name, the data type, and the order of the two parameters are the same in the task and in the service. In our example, this is the case with the *id* parameter. To facilitate the understanding of the examples of this section, the parameters that are the focus of attention of each one has been underlined:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

But at this point we also consider that two parameters, one from the task and one from the service, can be aligned if they are semantically compatible and their data types are also compatible, regardless of the order in which they appear in their respective signatures. Thus, in our approach, a task parameter can also be aligned when any of the following three Basic Incompatibilities occur (or any possible combination of them):

- **Basic Incompatibility 1.** Incompatibility of a pair of parameters in terms of the semantic similarity of their names. This incompatibility arises when the names of parameters that can be potentially aligned are not identical, but they might have a similar meaning. This incompatibility could be resolved using semantic mechanisms based on the identification of similar concepts (e.g., using synonyms). In our example, this case occurs with the *lastName* parameter of the task and the *surName* parameter of the service:

Task: ***Obtain User***(*in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))*

Service: ***getUser***(*in id (long), out gender (string), out surName (string), out age (short), out name (string))*

- **Basic Incompatibility 2.** Incompatibility of a pair of parameters in terms of their order in the parameter list. This incompatibility arises when the order of a pair of parameters that can be potentially aligned is not the same. Consequently, a mechanism to identify and to resolve this incompatibility should be defined. In our example, this case occurs with *firstName* and *gender* (and *sex* and *name*) parameters:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

- **Basic Incompatibility 3.** Incompatibility of a pair of parameters in terms of their data types. This incompatibility arises when the data type of a pair of parameters that can be potentially aligned is not the same. Nevertheless, different data types can be considered compatible in some cases. The data type that we consider compatible are listed in Table 3. The table gives only the basic compatible types; these could be extended to other domains. In these cases, it is possible to type-cast the two parameters. In our example, this case occurs with the *age* parameter:

TABLE 3. Compatible types.

Source	Target
short	int
int	float
float	double

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

C. COMPATIBILITY OF THE ENTIRE SIGNATURE

In the third and last stage, it is necessary to ensure that the entire task and service signatures can be aligned. We will consider that a task can be aligned in the following cases:

- **Alignment type 1 or Natural Alignment.** This case arises when the method names of the task and the service are compatible, each of the task's input parameters has a compatible input parameter in the service, each of the task's output parameters has an compatible output parameter in the service, and the service has no unaligned parameter. This is the case of our example:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

- **Alignment type 2.** In principle, a task and a service could be considered unalignable if the service includes extra parameters that cannot be aligned with the task. But in practice this is not a problem if every one of the task's parameters is aligned. The following example illustrates this situation:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string)), out address (string), out postCode (long))

Thus, although the service's *address* and *postCode* parameters cannot be aligned, they can be ignored because the task does not need them.

- **Alignment type 3.** This situation occurs when a service that can be aligned has a default value in some one of its input parameters with which no parameter of the task can be aligned. Again, there is no problem if every one of the task's parameters is aligned because the service's input parameter that is unaligned has a default value. The following example illustrates this situation:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string))

Service: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string), in requestDate(date) = SYSDATE)

Thus, the service's *requestDate* parameter can not be aligned, but this is not a problem due to the parameter's default value.

- **Alignment type 4 or Special Alignment.** This situation arises when a task is not completely aligned with any of the available services, but it could be implemented by means of several sequentially invoked services. The following example illustrates this situation:

Task: ***Obtain User***(in id (long), out firstName (string), out lastName (string), out age (int), out sex (string), out domicile (string), out postCode (long))

Service1: ***getUser***(in id (long), out gender (string), out surName (string), out age (short), out name (string))

Service2: ***obtainUserLocation***(in id (long), out address (string), out postCode (long), out phoneNumber (long))

Thus, the task's *domicile* and *postCode* parameters cannot be aligned with any of the *getUser* service's parameters. But the *obtainUserLocation* service has the same input parameter (*id*) and two parameters, *address* and *postCode*, that are compatible with *domicile* and *postCode*. An implementation in which there is invocation of first *getUser* and then *obtainUserLocation* would return all the values needed by the *obtain User* task.

IV. ALIGNMENT PROCESS AND INCOMPATIBILITY ANALYSIS: MATHEMATICAL FORMULATION

In this section we shall define and note the tasks and services and their parameters mathematically so as to give a clear view of exactly what we are doing at every step, and to be able to transfer this scheme to other situations.

We shall use T to denote a business process task, S a service, and P a parameter. A given parameter P will be either an input parameter IP or an output parameter OP. And a given input parameter IP will be either a mandatory input parameter MIP or a non-mandatory input parameter NIP.

TASKS

- The set of tasks $\mathcal{T}$ will be the following ordered sequence:

$$\mathcal{T} = \{T_i\}_{i=1}^{N} = \{T_1, \ldots, T_N\}.$$

For instance, $\mathcal{T}$ = *{Obtain User,...,Create Department}*.

- Given a task $T_i \in \mathcal{T}$, the set of input parameters $\mathcal{IP}(T_i)$ associated with this task T_i will be the following ordered sequence:

$$\mathcal{IP}(T_i) = \{IP_k(T_i)\}_{k=1}^{n_i} = \left\{IP_1(T_i), \ldots, IP_{n_i}(T_i)\right\}.$$

For instance, $\mathcal{IP}(Obtain\ User) = \{id(long)\}$.
Every input parameter IP of a task will be a mandatory parameter.
The number of input parameters for a task T_i is the cardinality of the set $\mathcal{IP}(T_i)$:

$$Card(\mathcal{IP}(T_i)) = |\mathcal{IP}(T_i)| = n_i.$$

For instance, $Card(\mathcal{IP}(Obtain\ User)) = 1$.

- Given a task $T_i \in \mathcal{T}$, the set of output parameters $\mathcal{OP}(T_i)$ associated with that task T_i will be the following ordered sequence:

$$\mathcal{OP}(T_i) = \{OP_l(T_i)\}_{l=1}^{n_i} = \left\{OP_1(T_i), \ldots, OP_{n^i}(T_i)\right\}.$$

For instance, $\mathcal{OP}(Obtain\ User)$ = *{firstName (string), lastName (string), age (int), sex (string)}*.
The number of output parameters for a task T_i is the cardinality of the set $\mathcal{OP}(T_i)$:

$$Card(\mathcal{OI}(T_i)) = |\mathcal{OP}(T_i)| = n^i.$$

For instance, $Card(\mathcal{OP}(Obtain\ User)) = 4$.

- For every $T_i \in \mathcal{T}$, the set of parameters of a task T_i is:

$$\mathcal{P}_{T_i} = \left\{\mathcal{IP}(T_i)) \bigcup \mathcal{OP}(T_i)\right\},$$

and therefore the total number of parameters of this task, T_i, will be the cardinality of this set

$$Card(\mathcal{P}_{T_i}) = |\mathcal{IP}T_i)| + |\mathcal{OI}T_i)| = n_i + n^i.$$

For instance, $\mathcal{P}(Obtain\ User)$ = *{id (long), firstName (string), lastName (string), age (int), sex (string)}* and $Card(\mathcal{P}(Obtain\ User)) = 5$.

SERVICES

- The set of services $\mathcal{S}$ will be the following ordered sequence:

$$\mathcal{S} = \{S_j\}_{j=1}^{M} = \{S_1, \ldots, S_M\}.$$

For instance, $\mathcal{S} = \{getUser, \ldots, getUserLocation\}$.

- Given a service $S_j \in \mathcal{S}$, the set of input parameters $\mathcal{IP}(S_j)$ associated with this serviceS_j will be the following ordered sequence:

$$\mathcal{IP}(S_j) = \{IP_r(S_j)\}_{r=1}^{m_j} = \{IP_1(S_j), \ldots, IP_{m_j}(S_j)\}.$$

For instance, $\mathcal{IP}(getUser) = \{id(long)\}$.
Every input parameter IP of a service will be a mandatory parameter.
The number of input parameters for a service S_j is the cardinality of the set $\mathcal{IP}(S_j)$:

$$Card(\mathcal{IP}(S_j)) = |\mathcal{IP}(S_j)| = m_j.$$

For instance, $Card(\mathcal{IP}(getUser)) = 1$.

- Given a service S_j, an input parameter can be a mandatory parameter $\mathcal{MIP}(S_j)$ or a non-mandatory parameter $\mathcal{NIP}(S_j)$:
The set of mandatory input parameters $\mathcal{MIP}(S_j)$ is the following ordered sequence:

$$\begin{aligned}\mathcal{MIP}(S_j) &= \{MIP_u(S_j)\}_{u=1}^{\overline{m}_j} \\ &= \{MIP_1(S_j), ..., MIP_{\overline{m}_j}(S_j)\}.\end{aligned}$$

The set of non-mandatory input parameters $\mathcal{NIP}(S_j)$ is the following ordered sequence:

$$\begin{aligned}\mathcal{NIP}(S_j) &= \{NIP_v(S_j)\}_{v=1}^{\underline{m}_j} \\ &= \{NIP_1(S_j), ..., NIP_{\underline{m}_j}(S_j)\}.\end{aligned}$$

Clearly $m_j = \overline{m}_j + \underline{m}_j$ and

$$\mathcal{IP}(S_j) = \mathcal{MIP}(S_j) \cup \mathcal{NIP}(S_j).$$

For instance, $\mathcal{IP}(getUser) = \{id\} \cup \{\} = \{id\}$.

- Given a service $S_j \in \mathcal{T}$, the set of output parameters $\mathcal{OP}(S_j)$ associated with this service S_j will be the following ordered sequence:

$$\begin{aligned}\mathcal{OP}(S_j) &= \{OP_s(S_j)\}_{s=1}^{m^j} \\ &= \{OP_1(S_j), \ldots, OP_{m^j}(S_j)\}.\end{aligned}$$

For instance, $\mathcal{OP}(getUser)$ = *{gender (string), surName (string), age (short), name(string)}*.
The number of output parameters for a service S_j is the cardinality of the set $\mathcal{OP}(S_j)$:

$$Card(\mathcal{OP}(S_j)) = |\mathcal{OP}(S_j)| = m^j.$$

For instance, $Card(\mathcal{OP}(getUser)) = 4$.

- For every $S_j \in \mathcal{S}$, the set of parameters of a service S_j is:

$$\mathcal{P}_{S_j} = \mathcal{IP}(S_j)) \cup \mathcal{OP}(S_j),$$

and therefore the total number of parameters of this service, S_j, will be the cardinality of this set

$$Card(\mathcal{P}_{S_j}) = |\mathcal{IP}(S_j)| + |\mathcal{OI}(S_j)| = m_j + m^j.$$

For instance, $\mathcal{P}(getUser)$ = *{id (long), gender (string), surName (string), age (short), name (string)}* and $Card(\mathcal{P}(getUser)) = 5$.

FUNCTIONS

In this subsection, we shall define different functions which will have the parameters defined above as variables. These functions will be useful during the alignment process:

- The function $Name(x)$ returns the name of a task, service, or parameter x.
- The function $Type(P)$ returns the type of a parameter P.
- Given a pair of parameters P_x, P_y, we shall say that these parameters are semantically compatible when their names are semantically similar. The function $SemP(P_x, P_y)$ returns true if the parameters P_x and P_y are semantically compatible.
- Given a pair of parameters P_x, P_y, we shall say that these parameters have compatible types when their data types are compatible. The function $CompType(P_x, P_y)$ returns true if the parameters P_x and P_y have compatible types.

ALIGNMENT

We shall define the Cartesian product $\mathcal{IP}(T) \times \mathcal{IP}(S)$ (writing $T \times S$ for simplicity) of the two sets T and S as the set of all ordered pairs (x, y) where $x \in \mathcal{IP}(T)$ and $y \in \mathcal{IP}(S)$. Given $(x, y) \in T \times S$, one has the projection functions $\Pi_1(T \times S) = \mathcal{IP}(T)$ and $\Pi_2(T \times S) = \mathcal{IP}(S)$.

A binary relation R between sets $\mathcal{IP}(T)$ and $\mathcal{IP}(S)$ is a subset of $\mathcal{IP}(T) \times \mathcal{IP}(S)$. Thus, a relation is a set of pairs. The interpretation of this subset is that it contains all the pairs for which the relation is true. We write xRy if the relation is true for x and y (equivalently, if $(x, y) \in R$).

Then:

- Given a task $T_i \in \mathcal{T}$ and a service $S_j \in \mathcal{S}$, a pair of aligning input parameters between T_i and S_j will be those semantically compatible and with compatible types:

$$\begin{aligned}\mathcal{AIP}(T_i, S_j) = &(IP_x(T_i), IP_y(S_j)) \\ &\bigwedge SemP(IP_x(T_i), IP_y(S_j)) \\ &\bigwedge CompType(IP_x(T_i), IP_y(S_j)).\end{aligned}$$

For instance, $\mathcal{AIP}(ObtainUser, getUser) = \{id\ (long), id\ (long)\}$.

- Given a task $T_i \in \mathcal{T}$ and a service $S_j \in \mathcal{S}$, the set of aligning input parameters $\mathcal{AIPTS}(T_i, S_j)$ between T_i and S_j will be the following ordered sequence:

$$\begin{aligned}\mathcal{AIPTS}(T_i, S_j) &= \{\mathcal{AIP}_z(T_i, S_j)\}_{z=1}^{Min(n_i, m_j)} \\ &= \{\mathcal{AIP}_1(T_i, S_j), \ldots, \mathcal{AIP}_{Min(n_i, m_j)}(T_i, S_j)\}.\end{aligned}$$

For instance, $\mathcal{AIPTS}(obtainUser, getUser)$ = *{{id (long) (0), id (long) (0)}}*.

- Given a task $T_i \in \mathcal{T}$ and a service $S_j \in \mathcal{S}$, a pair of aligning output parameters between T_i and S_j will be those semantically compatible and with compatible types:

$$\begin{aligned}\mathcal{AOP}(T_i, S_j) = &(OP_x(T_i), OP_y(S_j)) \\ &\bigwedge SemP(OP_x(T_i), OP_y(S_j)) \\ &\bigwedge CompType(OP_x(T_i), OP_y(S_j)).\end{aligned}$$

  For instance, $\mathcal{AOP}(ObtainUser, getUser) = \{sex\ (string), gender\ (string)\}$.
- Given a task $T_i \in \mathcal{T}$ and a service $S_j \in \mathcal{S}$, the set of aligning output parameters $\mathcal{AOPTS}(T_i, S_j)$ between T_i and S_j will be the following ordered sequence:

$$\begin{aligned}&\mathcal{AOPTS}(T_i, S_j) = \left\{\mathcal{AOP}_z(T_i, S_j)\right\}_{z=1}^{Min(n^i, m^j)} \\ &= \left\{\mathcal{AOP}_1(T_i, S_j), \ldots, \mathcal{AOP}_{Min(n^i, m^j)}(T_i, S_j)\right\}.\end{aligned}$$

  For instance, $\mathcal{AOPTS}(obtainUser, getUser) = \{\{firstName\ (string)\ (1), name(string)\ (2)\}, \{lastName\ (string)\ (2), surName\ (string)\ (1)\}, \{age(short)(3), age(int)(3)\}, \{sex(string)(4), gender(string)(4)\}\}$.

A. SEMI-AUTOMATIC SEMANTIC ALGORITHM

This subsection describes the semantic algorithm used to implement the alignment process between BPTs and Web services. This paper extends a semantic algorithm, explained in our previous works [20], [41], improving its efficiency by detecting and resolving incompatibilities. Nevertheless, with the aim of keeping the paper self-contained, we shall first describe the basic semantic algorithm, and then describe the incompatibility identification and resolution.

1) THE DOMAIN'S SEMANTIC DICTIONARY

A semantic dictionary was defined in order to establish a common semantic basis and obtain a better result in the alignment process. This dictionary defines the binding of a term with other terms related to it within a domain. The main advantage of using a semantic dictionary is that it helps developers and analysts to define BPMN models using a common language, and, as explained in [42], the use of a lexical semantic base also helps to find ambiguities or duplications in the definition of the business processes. To continue with the example, the *Obtain User* task could be a task in which the verb *Obtain* could be identified as a CRUD operation and *User* as an object.

A semantic dictionary is defined for a particular domain by business process analysts and domain experts. The structure of the semantic dictionary is a set of terms, and for each of these terms a set of terms related to it. The relations taken into account in the design of the dictionary are not only those of closeness of terms according to the domain but also synonymy, hyponymy, and meronymy, and such simple relations as abbreviated terms.

In the following, we shall describe the service discovery process and the semantic algorithm it uses.

2) SERVICE DISCOVERY PROCESS

The service discovery process uses the aforementioned semantic algorithm. The original definition of this algorithm, which is based on the James Z. Wang, Farha Ali, and Rashmy Appaneravanda algorithm [43] (henceforth, *Wang-Ali*), is as follows. First, an ontological similarity measure is defined based on *Tversky's* normalization formula model [44]. This model starts from the idea that common features tend to increase the perceived similarity of two concepts, whereas differences in features tend to diminish it [45]. The main objective of the original algorithm is to obtain a measure of the similarity of two ontologies. To simplify the set of operations, a set of synonyms (called a *senses set*) is used to summarize the semantics of the ontology.

If the Target Ontology senses set is TO and the Source Ontology senses set is SO, the difference between them, denoted by $TO - SO$, is defined as $TO - SO = \{x \mid x \in TO \wedge x \notin SO\}$. the semantic difference between the two ontologies is then calculated by the Wang-Ali algorithm as the function: $D(TO, SO) = |TO - SO|/|T|$, where $0 \leq D(TO, SO) \leq 1$.

Thus, this function is equal to *1* when there are no common elements between the TO and SO senses sets, i.e., $|TO - SO| = |TO|$. But if TO is a subset of SO ($TO \subseteq SO$), i.e., $|TO - SO| = 0$, then the function is equal to *0*.

We made some modifications to this algorithm to adapt it to the context and specific features of our project. First, we adapted the algorithm to compare sets of terms (the original Wang-Ali [43] applies to the comparison of complete ontologies). Thus, we do not compare our two complete sets of terms (in our case business process tasks and the services identified) but instead each task and its set of synonyms (as the first set of terms) with each of the services and its set of synonyms (as the second set of terms). By *set of synonyms of a term* we mean terms related to it in the semantic dictionary, using the relationships taken into account in that dictionary.

Second, since business processes tasks and services are normally identified by composite words (based on basic writing-task rules and reverse engineering processes, respectively), it is necessary to identify each word and apply a specific weight to it during execution of the algorithm. As an example, the *Obtain User* task could be split into two words: *Obtain* (action) and *User* (object). We therefore added the assignment of a configurable weight to the action identified or to the object on which the action is performed. Initially, the object will have more weight because the combination of synonymous actions on different objects are normally not considered to be synonymous. It should be noted that our algorithm takes into account not only the task or service name, but also the input and output parameters. As we have seen, the whole task or service name and its parameters is called *signature*.

If $\mathcal{T}$ is the set of business processes tasks and $\mathcal{S}$ the set of services, the algorithm is implemented as follows. For each task (T) stored in $\mathcal{T}$, each service (S) stored in $\mathcal{S}$ is checked, and a value of similarity between the task and each of the services is calculated. This similarity measure will be a value

between *0* and *1*, taking into account that the closer to *0* the value, the more similar the terms will be. Let T a task of the business process tasks set ($\mathcal{T}$) and S a service of the services set ($\mathcal{S}$). To do the division between action and object in each of these terms, we shall use *T1* to denote the action of the T task, and *T2* to denote the object of that task. The same for services. I.e., *S1* is the action of the S service, and *S2* is its object. The weights applied to action and object will be denoted by *weighta* and *weighto*, respectively.

The pseudocode of our modification of the semantic algorithm is shown in Algorithm 1. For simplicity, the second part of the algorithm that takes into account the task and service parameters in the alignment is not shown.

Algorithm 1 Semantic Algorithm for the Alignment Process

```
Wang-Ali-extended algorithm (T, S, weighta, weighto,
Wang-Ali-SimMatr[][])
begin
for each T ∈ T   //for each task
    T1 = action of T
    T2 = object of T
    Syn_T1 = set of synonyms of T1
    Syn_T2 = set of synonyms of T2
  for each S ∈ S   //for each service
        S1 = action of S
        S2 = object of S
        Syn_S1 = set of synonyms of S1
        Syn_S2 = set of synonyms of S2
        v1 = Syn_T1 - Syn_S1 / Syn_T1
            //v1: semantic diff. between actions
        v2 = Syn_T2 - Syn_S2 / Syn_T2
            //v2: semantic diff. between objects
        Wang-Ali-SimMatr[t][s] = v1 * weighta +
                                 v2 * weighto
    end for
end for
end
```

As one can see from the pseudocode, the result of Algorithm 1 is the table *Wang-Ali-SimMatr* of t rows (number of terms in the business process tasks set) and s columns (number of terms in the services set). The i-th row and j-th column of this matrix contains the value *Wang-Ali-SimMatrij=SimMatr1*(t_i, s_j), i.e., the index of similarity between each business process task and each service of the service layer identified previously.

B. IMPROVING THE SEMANTIC ALGORITHM: ALLOWING ADAPTATIONS

As has been indicated above, various incompatibilities related to the parameter list must be resolved in order to increase the percentage of business process tasks aligned suitably with the services available in the services layer. In this sense, in this subsection we shall describe mathematically how to identify and resolve these incompatibilities.

Taking into account the mathematical definitions of *tasks*, *services*, *functions* and *aligning* given above, in Table 4 we define a classification of a set of situations in which input and output parameters could be suitably aligned: Normal Input Alignment (NIA), Mandatory Input Alignment (MIA), Normal Output Alignment (NOA), Inclusive Output Alignment (IOA), and Composite Output Alignment (COA).

Table 5 defines the cases corresponding to the possible input and output parameter combinations. Each combination is to be treated as a special case. In the following, we shall describe each of these cases separately, including an example, its mathematical representation, and a rationale for how the incompatibility could be resolved.

It should be noted that the names of the tasks are shown with a single word from here on in order to designate them in the same way as the semantic algorithm does.

1) CASE 1: NIA X NOA

Consider the following task *obtainUser* and service *getUser*:

Task: ***obtainUser***(in id (long), out firstName (string), out lastName (string), out age(int), out sex (string))

Service: ***getUser***(in id (long), out surName (string), out name (string), out age(short), out gender (string))

Case 1 has the three types of basic incompatibilities: Basic Incompatibility 1 (or incompatibility of their names, between *firstName* and *name* parameters), Basic Incompatibility 2 (or incompatibility of their order, between *firstName* and *surName* parameters) and Basic Incompatibility 3 (or incompatibility of their data types, between *age* (*int* type) and *age* (*short* type) parameters).

With the mathematical formulation given above, the corresponding task and service definitions are as follows:

//Task definition
$\mathcal{T} = \{obtainUser, \dots\}$
$\mathcal{IP}(obtainUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(obtainUser)) = 1$
$\mathcal{OP}(obtainUser) = \{firstName\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$Card(\mathcal{OP}(obtainUser)) = 4$

//Service definition
$\mathcal{S} = \{getUser, \dots\}$
$\mathcal{MIP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{MIP}(getUser)) = 1$
$\mathcal{NIP}(getUser) = \{\}$
$Card(\mathcal{NIP}(getUser)) = 0$
$\mathcal{IP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(getUser)) = 1$
$\mathcal{OP}(getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\}$
$Card(\mathcal{OP}(getUser)) = 4$

TABLE 4. Alignments supported by the alignment process.

Input parameters	Output parameters
Normal Input Alignment (NIA): $\Pi_1(\mathcal{AIPTS}(T_i, S_j)) = \mathcal{IP}(T_i)$ $\wedge$ $\Pi_2(\mathcal{AIPTS}(T_i, S_j)) = \mathcal{IP}(S_j)$	Normal Output Alignment (NOA): $\Pi_1(\mathcal{AOPTS}(T_i, S_j)) = \mathcal{OP}(T_i)$ $\wedge$ $\Pi_2(\mathcal{AOPTS}(T_i, S_j)) = \mathcal{OP}(S_j)$
Mandatory Input Alignment (MIA): $\Pi_1(\mathcal{AIPTS}(T_i, S_j)) = \mathcal{IP}(T_i)$ $\wedge$ $\Pi_2(\mathcal{AIPTS}(T_i, S_j)) \supseteq \mathcal{MIP}(S_j)$	Inclusive Output Alignment (IOA): $\Pi_1(\mathcal{AOPTS}(T_i, S_j)) = \mathcal{OP}(T_i)$ $\wedge$ $\Pi_2(\mathcal{AOPTS}(T_i, S_j)) \subseteq \mathcal{OP}(S_j)$
	Composite Output Alignment (COA): $\Pi_1(\mathcal{AOPTS}(T_i, (S_1 \cup S_2 \cup \cdots \cup S_n))) = \mathcal{OP}(T_i)$ $\wedge$ $\Pi_2(\mathcal{AOPTS}(T_i, (S_1 \cup S_2 \cup \cdots \cup S_n))) \subseteq \mathcal{OP}(S_1) \cup \mathcal{OP}(S_2) \cup \cdots \cup \mathcal{OP}(S_n)$

TABLE 5. Cases suitable for treatment by the alignment process.

Input Alignment x Output Alignment	Case
NIA x NOA	1
NIA x IOA	2
MIA x NOA	3
MIA x IOA	4
NIA x COA or MIA x COA	5

The set of alignments obtained above to analyze individually the *task input parameters* and the *service input parameters* are:

$\mathcal{AIPTS}$(*obtainUser,getUser*) = {{*id(long)(0), id(long)(0)*}}

$\Pi_1(\mathcal{AIPTS})$(*obtainUser,getUser*) = {*id(long)*}

$\Pi_2(\mathcal{AIPTS})$(*obtainUser,getUser*) ={*id(long)*}

The set of alignments obtained above to analyse individually the *task output parameters* and the *service output parameters* are:

$\mathcal{AOPTS}$(*obtainUser,getUser*) = {{*firstName(string)(1), name(string)(2)*}, {*lastName (string)(2), surName(string)1)*}, {*age(short)(3), age(int)(3)*}, {*sex(string)(4), gender (string)(4)*}}

$\Pi_1(\mathcal{AOPTS})$(*obtainUser,getUser*) = {*firstName (string), lastName (string), age(int), sex (string)*}

$\Pi_2(\mathcal{AOPTS})$(*obtainUser,getUser*) = {*surName (string), name (string), age(short), gender (string)*}

The *input parameters* alignment satisfies NIA:

$\Pi_1(\mathcal{AIPTS}$(*obtainUser,getUser*) = $\mathcal{IP}$(*obtainUser*) = {*id(long)*}

$\wedge$

$\Pi_2(\mathcal{AIPTS}$(*obtainUser,getUser*) = $\mathcal{IP}$(*getUser*) = {*id(long)*}

And the *output parameters* alignment satisfies NOA:

$\Pi_1(\mathcal{AOPTS}$(*obtainUser,getUser*) = $\mathcal{OP}$(*obtainUser*) = {*firstName (string), lastName (string), age(int), sex (string)*}

$\wedge$

$\Pi_2(\mathcal{AOPTS}$(*obtainUser,getUser*) = $\mathcal{OP}$(*getUser*) = {*surName (string), name (string), age(short), gender (string)*}

The complete alignment set (see Fig. 3) is the following:

$\mathcal{APTS}$(*obtainUser,getUser*)={{*id(long)(0), id(long)(0)*}, {*firstName(string)(1), name(string)(2)*}, {*lastName (string)(2), surName(string)(1)*}, {*age(short)(3),age(int)(3)*}, {*sex(string)(4), gender(string)(4)*}}

Resolution of the incompatibility

This incompatibility only includes the parameter names, their order, or their data types, i.e., it only includes the Basic Incompatibilities (as defined above). Consequently, the

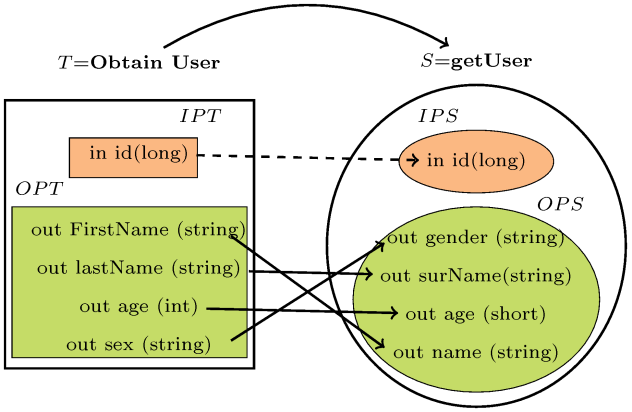

FIGURE 3. Case 1 example. NIA x IOA.

obtainUser task can invoke the *getUser* service taking into account $\mathcal{AIPTS}$ and $\mathcal{AOPTS}$.

In order to solve this incompatibility, a wrapper of the task is automatically generated. Thus, Algorithm 2 shows this wrapper which allows the Web service (*getUser*) to be invoked from the business process task (*obtainUser*). Note that this wrapper describes how to align the task with respect to the services identified previously.

Algorithm 2 Wrapper Example to Resolve the Case 1 Incompatibility Type

```
Task obtainUser(in id (long), out firstName (
    string), out lastName (string), out age(int),
    out sex (string))

begin
        tmp_age: short  // stage (a)

        //getUser(in id (long), out surName (
            string), out name (string), out short
            age, out gender (string))
        invoke getUser( id, lastName, firstName,
            tmp_age, sex)  // stage (b)

        age = (int)tmp_age  // stage (c)
end
```

Each incompatibility case can be resolved using the generated wrappers. Thus, the process of generation of these wrappers is composed of the following stages: a.

1) Identification of the parameters that do not have the same type in the task and the service or that have a different position in the parameter list or that are not aligned. Then, if necessary, temporary variables that allow the correct invocation of the service are declared. For this, $\mathcal{APTS}$(*obtainUser,getUser*)= {{id(long)(0), id(long)(0)}, {firstName(string)(1), name(string)(2)}, {lastName(string)(2), surName(string)(1)}, {age (short)(3), age(int)(3)}, {sex(string)(4), gender (string)(4)}} is analyzed.
2) Invocation (**invoke**) of the service by means of the temporary variables defined.
3) Assignation of the values returned by the service in the output parameters to the related parameters.

In Case 1, at stage (a), a temporary variable *tmp_age* of *short* type is defined in order to invoke the service. At stage (c) the value returned by the service in the *tmp_age* parameter is assigned (using casting) to the *int age* parameter.

2) CASE 2: NIA X IOA

Consider the following task *obtainUser* and service *getUser*:

Task: ***obtainUser***(in id (long), out firstName (string), out lastName (string), out age(int) , out sex (string))

Service: ***getUser***(in id (long), out surName (string), out name (string), out age(short), out gender (string), address (string), postCode (long))

//Task definition
$\mathcal{T} = \{obtainUser, \dots\}$
$\mathcal{IP}(obtainUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(obtainUser)) = 1$
$\mathcal{OP}(obtainUser) = \{firstName\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$Card(\mathcal{OP}(obtainUser)) = 4$

//Service definition
$\mathcal{S} = \{getUser, \dots\}$
$\mathcal{MIP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{MIP}(getUser)) = 1$
$\mathcal{NIP}(getUser) = \{\}$
$Card(\mathcal{NIP}(getUser)) = 0$
$\mathcal{IP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(getUser)) = 1$
$\mathcal{OP}(getUser) = \{surName(string),\ name\ (string),\ age(short),\ gender\ (string),address\ (string),\ postCode\ (long)\}$
$Card(\mathcal{OP}(getUser)) = 6$

The set of alignments obtained above to analyse individually the *task input parameters* and the *service input parameters* are:

$\mathcal{AIPTS}(obtainUser,getUser) = \{\{id(long)(0), id(long)(0)\}\}$
$\Pi_1(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$
$\Pi_2(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$

The set of alignments obtained above to analyse individually the *task output parameters* and the *service output parameters* are:

$\mathcal{AOPTS}(obtainUser,getUser) = \{\{firstName(string)(1), name(string)(2)\}, \{lastName(string)(2), surName(string)1)\}, \{age(short)(3), age(int)(3)\},\{sex(string)(4),gender(string)(4)\}$
$\Pi_1(\mathcal{AOPTS})(obtainUser,getUser) = \{firstName\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$\Pi_2(\mathcal{AOPTS})(obtainUser,getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\}$

The *input parameters* alignment satisfies NIA:

$\Pi_1(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(obtainUser) = \{id(long)\}$
$\wedge$
$\Pi_2(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(getUser) = \{id(long)\}$

However, the *output parameters alignment* does not satisfy NOA:

$\Pi_1(\mathcal{AOPTS}(obtainUser,getUser) = \mathcal{OP}(obtainUser) = \{firstName\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$\wedge$
$\Pi_2(\mathcal{AOPTS}(obtainUser,getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\} \neq \mathcal{OP}(getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string),address\ (string),\ postCode\ (long)\}$

But the *output parameters alignment* does satisfy IOA:

$\Pi_1(\mathcal{AOPTS}(obtainUser,getUser) = \mathcal{OP}(obtainUser) = \{firstName\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$\wedge$
$\Pi_2(\mathcal{AOPTS}(obtainUser,getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\} \subseteq \mathcal{OP}(getUser) =$

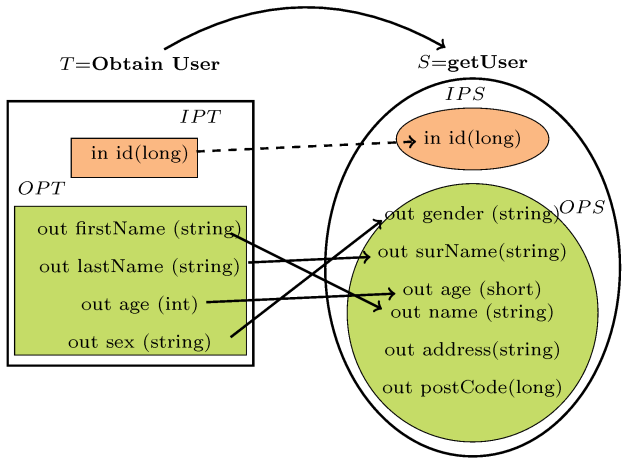

FIGURE 4. Case 2 example. NIA x IOA. Task and service input parameters aligned, and task output parameters aligned.

{*surName (string), name (string), age(short), gender (string), address (string), postCode (long)*}

The complete alignment set (see Fig. 4) is the following:

$\mathcal{APTS}$(*obtainUser,getUser*)={{*id(long)(0), id(long)(0)*}, {*firstName(string)(1), name (string)(2)*}, {*lastName (string)(2), surName(string)(1)*},{*age(short)(3), age(int)(3)*}, {*sex(string)(4), gender(string)(4)*}

Resolution of the incompatibility

Besides the resolution of the compatibility based on the Basic Incompatibility (parameter names, parameter order, or compatible data types), a wrapper which allows temporary variables to be defined should itself be defined to resolve the current incompatibility. These temporary variables will be used in order to handle the Web service output parameters which are not aligned with the output parameter task.

Next, a pseudocode fragment (Algorithm 3) is generated to implement a wrapper which allows the Web service (*getUser)* to be invoked from the business process task (*obtainUser*). Note that this wrapper describes how to align the task with respect to the services identified previously.

In Case 2, at stage (a), three temporary variables are defined, one for the invocation with different data type (*tmp_age*) and, two additional variables that make possible the invocation of the service with two output parameters that the task does not initially have (*tmp_adress* and *tmp_postCode*). Stage (b) invokes the service. Finally, at stage (c), the value returned by the service in the *tmp_age* parameter is assigned (using casting) to the *int age* parameter.

3) CASE 3. MIA X NOA

Consider the following task *obtainUser* and service *getUser*:

Task: ***obtainUser***(in id (long), out firstName (string), out lastName (string), out age(int) , out sex (string))

Algorithm 3 Wrapper Example to Resolve the Case 2 Incompatibility Type

```
Task obtainUser(in id(long), out firstName(string)
    , out lastName(string), out age(int), out sex(
    string))

begin
        tmp_age:short     // stage (a)
        tmp_adress: String
        tmp_postCode: long

        //getUser(in id (long), out surName (
            string), out name (string), out age(
            short), out gender( string), out
            address(string), out postCode(long))
        invoke getUser( id, lastName, firstName,
            tmp_age, sex, tmp_address,
            tmp_postCode)    // stage (b)

        age= (int)tmp_age  // stage (c)
end
```

Service: ***getUser***(in id (long), out surName (string), out name (string), out age(short), out gender (string), ***in requestDate(date)=SYSDATE***)

As one observes, the *getUser* service requires a special input parameter denoted *requestDate* which indicates a specific date on which the data should be queried (e.g., it could be used to obtain the user's address on a certain date). Note that this parameter has SYSDATE as its default value if the Web service invocation does not include a value for it.

// Task definition
$\mathcal{T} = \{obtainUser, \dots\}$
$\mathcal{IP}(obtainUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(obtainUser)) = 1$
$\mathcal{OP}(obtainUser) = \{firstName(string),\ lastName(string),\ age(int),\ sex\ (string)\}$
$Card(\mathcal{OP}(obtainUser)) = 4$

//Service definition
$\mathcal{S} = \{getUser, \dots\}$
$\mathcal{MIP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{MIP}(getUser)) = 1$
$\mathcal{NIP}(getUser) = \{requestDate(date)=SYSDATE\}$
$Card(\mathcal{NIP}(getUser)) = 1$
$\mathcal{IP}(getUser) = \{id\ (long),\ requestDate(date)=SYSDATE\}$
$Card(\mathcal{IP}(getUser)) = 2$
$\mathcal{OP}(getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\}$
$Card(\mathcal{OP}(getUser)) = 4$

The set of alignments obtained above to individually analyse the *task input parameters* and the *service input parameters* are:

$\mathcal{AIPTS}(obtainUser,getUser) = \{\{id(long)(0),id(long)(0)\}\}$
$\Pi_1(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$
$\Pi_2(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$

The set of alignments obtained above to individually analyse the *task output parameters* and the *service output parameters* are:

$\mathcal{AOPTS}$(*obtainUser,getUser*) = {{*firstName(string)(1), name (string)(2)*}, {*lastName (string)(2),surName (string)1)*}, {*age (short)(3), age (int)(3)*}, {*sex (string)(4), gender (string)(4)*}

$\Pi_1(\mathcal{AOPTS})$(*obtainUser,getUser*) = {*firstName (string), lastName (string), age(int), sex (string)*}

$\Pi_2(\mathcal{AOPTS})$(*obtainUser,getUser*) = {*surName (string), name (string), age(short), gender (string)*}

The *input parameters alignment* does not satisfy NIA:

$\Pi_1(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(obtainUser)$ = {*id(long)*}

$\wedge$

$\Pi_2(\mathcal{AIPTS}(obtainUser,getUser)$ = {*id(long)*} $\neq$ $\mathcal{IP}$ (*getUser*) = {*id (long),requestDate(date)=SYSDATE*}

However, the *input parameters alignment* does satisfy MIA:

$\Pi_1(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(obtainUser)$ = {*id(long)*}

$\wedge$

$\mathcal{AIPTS}$(*obtainUser,getUser*) = {*id(long)*} $\supseteq$ $\mathcal{MIP}$ (*getUser*) = {*id (long)*}

And the *output parameters alignment* satisfies NOA:

$\Pi_1(\mathcal{AOPTS}(obtainUser, getUser) = \mathcal{OP}(obtainUser)$ = {*firstName (string), lastName (string), age(int), sex (string)*}

$\wedge$

$\Pi_2(\mathcal{AOPTS}(obtainUser,getUser) = \mathcal{OP}(getUser)$ = {*surName (string), name (string), age(short), gender (string)*}

The complete alignment is (see Fig. 5):

$\mathcal{APTS}$(*obtainUser, getUser*) = {{*id(long) (0), id(long) (0)*}, {*firstName(string) (1), name (string) (2)*}, {*lastName(string) (2), surName(string) (1)*}, {*age(short) (3), age(int) (3)*}, {*sex(string) (4), gender(string) (4)*}

Resolution of the incompatibility

This alignment could be carried out by applying the resolution of the incompatibility described for Case 1. This is because in this case it is possible to ignore the non-aligned input parameters allowing them a default value in the Web service.

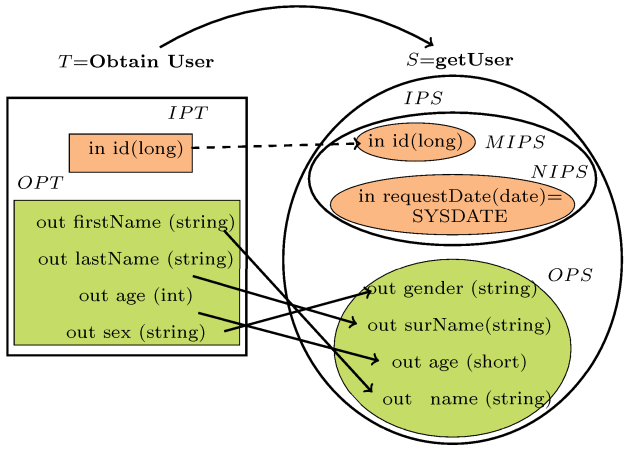

FIGURE 5. Case 3 example. MIA x NOA. Task input parameters aligned, and extra service input parameters are default values. Task output parameters are also aligned.

Next, a pseudocode fragment (Algorithm 4) is generated to implement a wrapper which allows the Web service (*getUser)* to be invoked from the business process task (*obtainUser*). Note that this wrapper describes how to align the task with respect to the services identified previously.

In Case 3, as happened in Case 1, at stage (a), a temporary variable *tmp_age* of *short* type is defined to make possible the invocation of the service at stage (b). At stage (c) the value returned by the service in the *tmp_age* parameter is assigned (using casting) to the *int age* parameter. No auxiliary variable is created for *requestDate* parameter because it has a default value.

Algorithm 4 Wrapper Example to Resolve the Case 3 Incompatibility Type

```
Task obtainUser(in id (long), out firstName (
    string), out lastName (string), out age(int),
    out sex (string))

begin
        tmp_age: short   // stage (a)

        //getUser(in id (long), out surName (
            string), out name (string), out age(
            short), out gender (string), in
            requestDate(date)=SYSDATE)
        invoke getUser( id, lastName, firstName,
            tmp_age, sex)  // stage (b)

        age= (int)tmp_age  // stage (c)
end
```

4) CASE 4. MIA X IOA

Consider the following task *obtainUser* and service *getUser*:

Task: ***obtainUser***(in id (long), out firstName (string), out lastName (string), out age(int) , out sex (string))

Service: ***getUser***(in id (long), out surName (string), out name (string), out age(short), out gender (string), address (string), postCode (long), ***in requestDate(date)=SYSDATE***)

As one observes, this example is a combination of Cases 2 and 3. I.e., on the one hand, there are service input parameters with default values which are not aligned (*requestDate*), and, on the other, there are service output parameters which could not be aligned (*address* and *postCode*).

// Task definition

$\mathcal{T}$ = {*obtainUser*, ...}

$\mathcal{IP}(obtainUser)$ = {*id (long)*}

$Card(\mathcal{IP}(obtainUser)) = 1$

$\mathcal{OP}(obtainUser)$ = {*firstName (string), lastName (string), age(int), sex (string)*}

$Card(\mathcal{OP}(obtainUser)) = 4$

//Service definition

$\mathcal{S}$ = {*getUser*, ...}

$\mathcal{MIP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{MIP}(getUser)) = 1$
$\mathcal{NIP}(getUser) = \{requestDate(date) = SYSDATE\}$
$Card(\mathcal{NIP}(getUser)) = 1$
$\mathcal{IP}(getUser) = \{id\ (long), requestDate(date) = SYSDATE\}$
$Card(\mathcal{IP}(getUser)) = 2$
$\mathcal{OP}(getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string), address\ (string), postCode\ (long)\}$
$Card(\mathcal{OP}(getUser)) = 6$

The set of alignments obtained above to analyse individually the *task input parameters* and the *service input parameters* are:

$\mathcal{AIPTS}(obtainUser,getUser) = \{\{id(long)(0), id(long)(0)\}\}$
$\Pi_1(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$
$\Pi_2(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$

The set of alignments obtained above to analyse individually the *task output parameters* and the *service output parameters* are:

$\mathcal{AOPTS}(obtainUser,getUser) = \{\{firstName\ (string)(1), name\ (string)(2)\}, \{lastName\ (string)(2), surName\ (string)1)\}, \{age\ (short)(3), age\ (int)(3)\}, \{sex\ (string)(4), gender\ (string)(4)\}\}$

$\Pi_1(\mathcal{AOPTS})(obtainUser,getUser) = \{firstName\ (string), lastName\ (string), age(int), sex\ (string)\}$

$\Pi_2(\mathcal{AOPTS})(obtainUser,getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string)\}$

The *input parameters* alignment does not satisfy NIA:

$\Pi_1(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(obtainUser) = \{id(long)\}$
$\wedge$
$\Pi_2(\mathcal{AIPTS}(obtainUser,getUser) = \{id(long)\} \neq \mathcal{IP}(getUser) = \{id\ (long), requestDate(date)=SYSDATE\}$

However, the *input parameters* alignment satisfies MIA:

$\Pi_1(\mathcal{AIPTS}(obtainUser,getUser) = \mathcal{IP}(obtainUser) = \{id(long)\}$
$\wedge$
$\Pi_2(\mathcal{AIPTS}(obtainUser,getUser) = \{id(long)\} \supseteq \mathcal{MIP}(getUser) = \{id\ (long)\}$

The *output parameters* alignment does not satisfy NOA:

$\Pi_1(\mathcal{AOPTS}(obtainUser,getUser) = \mathcal{OP}(obtainUser) = \{firstName\ (string), lastName\ (string), age(int), sex\ (string)\}$
$\wedge$
$\Pi_2(\mathcal{AOPTS}(obtainUser,getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string)\} \neq \mathcal{OP}(getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string), address\ (string), postCode\ (long)\}$

But the *output parameters* alignment does satisfy IOA:

$\Pi_1(\mathcal{AOPTS}(obtainUser,getUser) = \mathcal{OP}(obtainUser) = \{firstName\ (string), lastName\ (string), age(int), sex\ (string)\}$
$\wedge$
$\Pi_2(\mathcal{AOPTS}(obtainUser,getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string)\} \subseteq \mathcal{OP}(getUser) = \{surName\ (string), name\ (string), age(short), gender\ (string), address\ (string), postCode\ (long)\}$

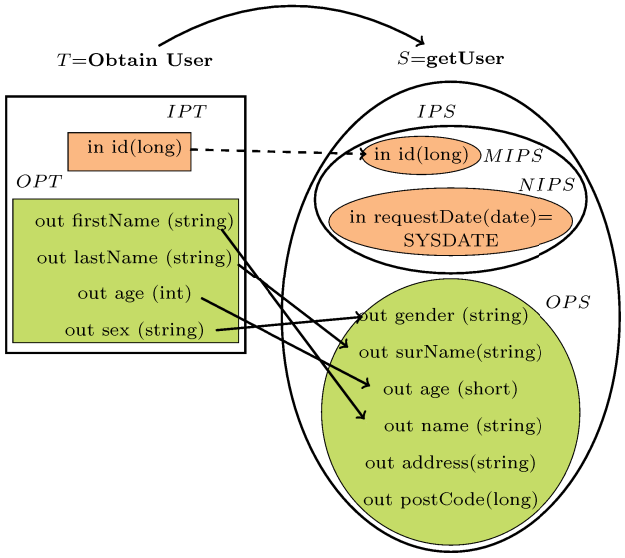

FIGURE 6. Case 4 example. MIA x IOA. Task input parameters aligned, and extra service input parameters are default values. Task output parameters are aligned although there exist extra service output parameters.

The complete set of alignments (see Fig. 6) is the following:

$\mathcal{APTS}(obtainUser,getUser) = \{\{id(long)(0), id(long)(0)\}, \{firstName(string)(1), name\ (string)(2)\}, \{lastName\ (string)(2), surName\ (string)(1)\}, \{age\ (short)(3), age\ (int)(3)\}, \{sex\ (string)(4), gender\ (string)(4)\}\}$

Resolution of the incompatibility

This alignment could be resolved by applying, on the one hand, the solution of the incompatibility described in Case 3, in which the input parameters with default values could not be aligned, or, on the other hand, the solution of the non-aligned service output parameters described in Case 2.

Next, a pseudocode fragment (Algorithm 5) is generated to implement a wrapper which allows the Web service (*getUser*) to be invoked from the business processes task (*obtainUser*). Note that this wrapper describes how to align the task with respect to the services identified previously.

In Case 4, as happened in previous cases, at stage (a), temporary variables (*tmp_age*, *tmp_adress* and *tmp_postCode*) are defined to make possible the invocation of the service at stage (b). At stage (c) the value returned by the service in the *tmp_age* parameter is assigned (by casting) to the *int age* parameter. Again, no auxiliary variable is created for *requestDate* parameter because it has a default value.

5) CASE 5. NIA X COA OR MIA X COA. NON-ALIGNED TASKS WHICH COULD BE ALIGNED BY INVOKING VARIOUS WEB SERVICES

One might find a special situation when a task should be resolved by means of invoking several services. It represents an opportunity to align tasks in a second round of our alignment process. Thus, considering a task not aligned with

Algorithm 5 Wrapper Example to Resolve the Case 4 Incompatibility Type

```
Task obtainUser(in id(long), out name(string), out
     lastName(string), out age(int), out sex (
    string))

begin
        tmp_age:short  //stage (a)
        tmp_adress: string
        tmp_postCode: long

        //getUser(in id (long), out surName (
            string), out name (string), out age(
            short), out gender( string), out
            address(string), out postCode(long),in
             requestDate(date)=SYSDATE))
        invoke getUser( id, lastName, firstname,
            tmp_age, sex, tmp_address,
            tmp_postCode)   //stage (b)

        age = (int)tmp_age  //stage (c)

end
```

any service, it could well be resolved by invoking various services.

In order to carry out this second round, firstly we look for a set of candidate services for each non-aligned task. Note that each candidate service should satisfy NIA or MIA with respect to the task. The mathematical definition for a successful alignment is:

$(\Pi_1(\mathcal{AIPTS}(T_i, S_j)) = \mathcal{IP}(T_i)(\forall S_j \in (S_1, S_2, \ldots, S_n)$
$\wedge$
$\Pi_2(\mathcal{AIPTS}(T_i, S_j)) = \mathcal{IP}(S_j)(\forall S_j \in (S_1, S_2, \ldots, S_n))$
$\wedge$
$(\Pi_1(\mathcal{AOPTS}(T_i, (S_1 \cup S_2 \cup \cdots \cup S_n))) = \mathcal{OP}(T_i)$
$\wedge$
$\Pi_2(\mathcal{AOPTS}(T_i, (S_1 \cup S_2 \cup \cdots \cup S_n))) \subseteq \mathcal{OP}(S_1) \cup \mathcal{OP}(S_2) \cup \cdots \cup \mathcal{OP}(S_n))$

where:

- T_i is a non-aligned task in the service discovery process.
- $ASL(T_i)$ (Align Service List) is a temporary list to store the partial alignment solutions. Each element in this list consists of a pair of aligned parameters.
- *CSL* (Candidate Service List) is a list which includes all the services that satisfy NIA or MIA with respect to T_i.

In order to illustrate this situation, consider an example with the *obtainUser* task and the *getUser* and *obtainUserLocation* services:

Task: ***obtainUser***(in id (long), out firstname (string), out lastName (string), out age(int) , out sex (string), *out domicile (string), out postCode (long)*)

Service1: ***getUser***(in id (long), out surName (string), out name (string), out age(short), out gender (string))

Service2: ***obtainUserLocation***(in id (long), out address (string), out postCode (long), out phoneNumber (long))

As one observes, the invocation of the two services (*getUser* and *obtainUserLocation*) allows all the information required by the *obtainUser* task to be obtained. This could be defined formally as follows:

// Task definition
$\mathcal{T} = \{obtainUser, \ldots\}$
$\mathcal{IP}(obtainUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(obtainUser)) = 1$
$\mathcal{OP}(obtainUser) = \{firstname\ (string),\ lastName\ (string),\ age(int),\ sex\ (string),\ domicile\ (string),\ postCode\ (long)\}$
$Card(\mathcal{OP}(obtainUser)) = 6$

//Services definition
$\mathcal{S} = \{getUser, obtainUserLocation \ldots\}$
$\mathcal{MIP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{MIP}(getUser)) = 1$
$\mathcal{NIP}(getUser) = \{\}$
$Card(\mathcal{NIP}(getUser)) = 0$
$\mathcal{IP}(getUser) = \{id\ (long)\}$
$Card(\mathcal{IP}(getUser)) = 1$
$\mathcal{OP}(getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\}$
$Card(\mathcal{OP}(getUser)) = 4$

$\mathcal{MIP}(obtainUserLocation) = \{id\ (long)\}$
$Card(\mathcal{MIP}(obtainUserLocation)) = 1$
$\mathcal{NIP}(obtainUserLocation) = \{\}$
$Card(\mathcal{NIP}(obtainUserLocation)) = 0$
$\mathcal{IP}(obtainUserLocation) = \{id\ (long)\}$
$Card(\mathcal{IP}(obtainUserLocation)) = 1$
$\mathcal{OP}(obtainUserLocation) = \{address\ (string),\ postCode\ (long),\ phoneNumber\ (long)\}$
$Card(\mathcal{OP}(obtainUserLocation)) = 3$

The set of alignments obtained above to analyse individually the *obtainUser* task parameters and the *getUser* service parameters are:

$\mathcal{AIPTS}(obtainUser,getUser) = \{\{id(long)(0),\ id(long)(0)\}\}$
$\Pi_1(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$
$\Pi_2(\mathcal{AIPTS})(obtainUser,getUser) = \{id(long)\}$
$\mathcal{AOPTS}(obtainUser,getUser) = \{\{firstname\ (string)(1),\ name\ (string)(2)\},\{lastName\ (string)(2),surName\ (string)(1)\},\ \{age\ (short)(3),\ age\ (int)(3)\},\ \{sex\ (string)(4),\ gender\ (string)(4)\}\}$
$\Pi_1(\mathcal{AOPTS})(obtainUser,getUser) = \{firstname\ (string),\ lastName\ (string),\ age(int),\ sex\ (string)\}$
$\Pi_2(\mathcal{AOPTS})(obtainUser,getUser) = \{surName\ (string),\ name\ (string),\ age(short),\ gender\ (string)\}$

The set of alignments obtained above to analyse individually the *obtainUser* task and the *obtainUserLocation* service parameters are:

$\mathcal{AIPTS}(obtainUser,obtainUserLocation) = \{\{id(long)(0),\ id(long)(0)\}\}$
$\Pi_1(\mathcal{AIPTS})(obtainUser,obtainUserLocation) = \{id(long)\}$
$\Pi_2(\mathcal{AIPTS})(obtainUser,obtainUserLocation) = \{id(long)\}$

$\mathcal{AOPTS}$(obtainUser,obtainUserLocation) = {{domicile (string)(5), address (string)(1)}, {postCode (long)(6), postCode (long)(2)}}

$\Pi_1(\mathcal{AOPTS})$(obtainUser,obtainUserLocation) = {domicile (string), postCode (long)}

$\Pi_2(\mathcal{AOPTS})$(obtainUser,obtainUserLocation) = {address (string), postCode(long)}

So, the *input parameters* alignment satisfies NIA with respect to each service analyzed:

$\Pi_1(\mathcal{AIPTS}$(obtainUser,getUser) = $\mathcal{IP}$(obtainUser) = {id(long)}

$\wedge$

$\Pi_2(\mathcal{AIPTS}$(obtainUser,getUser) = $\mathcal{IP}$(getUser) = {id(long)}

$\wedge$

$\Pi_1(\mathcal{AIPTS}$(obtainUser,obtainUserLocation) = $\mathcal{IP}$(obtainUser) = {id(long)}

$\wedge$

$\Pi_2(\mathcal{AIPTS}$(obtainUser,obtainUserLocation) = $\mathcal{IP}$(obtainUserLocation) = {id(long)}

The *output parameters* alignment do not satisfy NOA and IOA with respect to each service individuality analyzed.

$\Pi_1(\mathcal{AOPTS}$(obtainUser,getUser) ={firstname (string), lastName (string), age(int), sex (string)} $\neq$ $\mathcal{OP}$(obtainUser) = {firstname (string), lastName (string), age(int), sex (string),domicile (string), out postCode (long)}

$\Pi_1(\mathcal{AOPTS}$(obtainUser,obtainUserLocation) = {domicile (string), postCode (long)} $\neq$ $\mathcal{OP}$(obtainUser) = {firstname (string), lastName (string), age(int), sex (string),domicile (string), out postCode (long)}

However, with both services together, the union of their output parameters allows NOA or IOA to be satisfied:

$\Pi_1(\mathcal{AOPTS}$(obtainUser,(getUser U obtainUserLocation)) = = $\mathcal{OP}$(obtainUser) = {firstname (string), lastName (string), age(int), sex (string),domicile (string), out postCode (long)}

$\wedge$

$\Pi_2(\mathcal{AOPTS}$(obtainUser,(getUser U obtainUserLocation)) = {surName (string), name (string), age(short), gender (string), domicile (string), postCode (long)}$\subseteq$ $\mathcal{OP}$(getUser U obtainUserLocation) = {surName (string), name (string), age(short), gender (string), domicile (string), postCode (long), phoneNumber (long)}

The complete set of alignments (see Fig. 7) is the following:

$\mathcal{APTS}$(obtainUser, (getUser $\cup$ obtainUserLocation)) = $\mathcal{APTS}$(obtainUser, getUser) $\cup$ $\mathcal{APTS}$(obtainUser, obtainUserLocation)

= {{id(long)(0), id(long)(0)}, {firstname(string)(1), name (string)(2)}, {lastName (string)(2), surName (string)(1)}, {ag e(short)(3), age (int)(3)}, {sex (string)(4), gender (string)(4)}}

$\cup$

{{id(long)(0), id(long)(0)}, {domicile (string)(5), address (string)(1)}, {postCode (long)(6), postCode (long)(2)}}

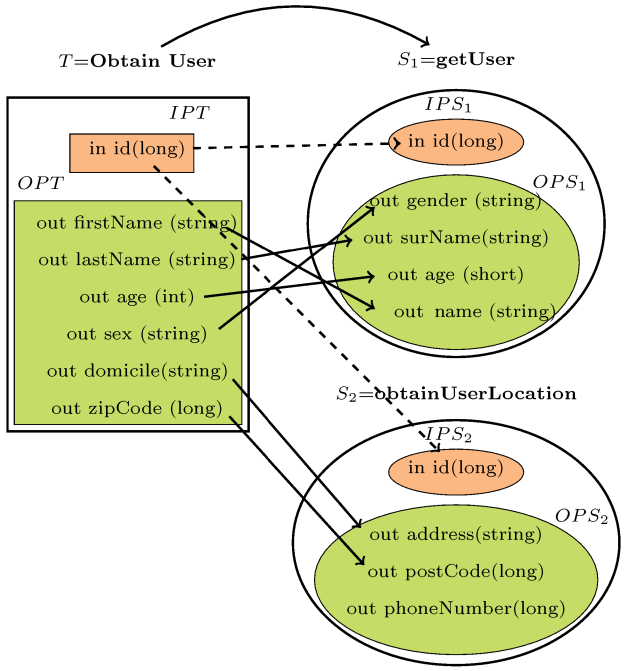

FIGURE 7. Case 5 example. NIA x COA or MIA x COA. Special case example.

The algorithm to carry out this second round includes the selection of the task not aligned previously. The output for each task includes the list of services that should be invoked to align each task. Then Algorithm (Algorithm 6) is applied (Fig. 7).

Resolution of the incompatibility

In order to align the *obtainUser* task with the *getUser* and *obtainUserLocation* services, a wrapper could be defined invoking the two services.

Next, a pseudocode fragment (Algorithm 7) is generated to implement a wrapper which allows the Web services (*getUser* and *obtainUserLocation*) to be invoked from the business process task (*obtainUser*). Note that this wrapper describes how to align the task with respect to the services identified previously.

In this case, at stage (a), temporary variables (*tmp_age* and *tmp_phoneNumber*) are defined to make possible the invocation of the services at stage (b). At stage (c) only the value returned by the first service in the *tmp_age* parameter is assigned (using casting) to the *int age* parameter.

It should be noted that the invocations to both services will be carried out in parallel, not sequentially. That is, one service does not complement the other.

V. VALIDATION AND DISCUSSION

In this section, we shall describe the process of validation which we applied to our approach. This process employed the case study presented above. It was divided into three steps: first, the terms defined in the dictionary and used in the case study were validated; second, a group of experts manually aligned the tasks and services of the case study;

Algorithm 6 Algorithm to Search for a Set of Services That Can be Aligned With Each Non-Aligned Task. The Result Is a Sequence of Service Invocations

```
Output_Model DiscoveryProcessExtended
BEGIN
        FOR EACH task T_i not aligned DO
                CLS = Candidate List of Services Accomplishing NIA or MIA for task T_i
                //Backtracking algorithm to search a T_i
                ASL = {}
                IF solve(T_i, CLS, ASL) = True THEN
                        //Add T_i and ASL to the Align Extended Matrix
                        //Add solution for T_i to the output model

                ENDIF
        ENDFOR
END

// T - Task
// CSL - Candidate Service List
// ASL - Aligned Service List
BOOLEAN solve(T, CLS, ASL)
BEGIN
        BOOLEAN solution = false
        IF  OPT(T) is included in (Proyection2(ASL)) THEN  //Including NOA and IOA
                RETURN True
        ELSE
                WHILE (NOT CLS.empty() AND NOT solution) DO
                        S=CLS.choose()  //Choose a service
                        ASL.add(S)  //Adding the service chosen as part of a possible solution
                        IF solve(T, CSL, ASL) == True THEN
                                solution = true
                        ELSE
                                ASL.remove (S)
                        ENDIF
                ENDWHILE
        RETURN solution;
END
```

and third, our approach was applied to the case study, and its results were compared with those of the group of experts (the semantic dictionary and the validation performed can be consulted in.[2]) This group of experts is composed of analysts of a technological park in the environment of the University and is composed of technology-based companies in collaboration with the University.

A. DICTIONARY VALIDATION

In the first step, we followed the validation procedure proposed by Cáceres *et al.* [46] to validate the terms of the dictionary that was constructed for the use in the case study. Our application of this procedure, which allows us to determine the degree of quality of the terms selection and the sets of synonyms of the defined semantic dictionary, consists of the following stages:

1) Definition of a first version of the research instrument, hereafter the semantic dictionary, by experts from the domain.
2) Selection of the group of experts. Concretely, the experts that were selected for the validation of the semantic dictionary were a group of twenty software developers and university researchers with previous experience in the domain.

[2]https://sites.google.com/site/migrasoa/uex-case-study

Algorithm 7 Wrapper Example to Resolve the Special Case Incompatibility Type

```
Task obtainUser(in id (long), out firstname (
    string), out lastName (string), out age(int),
    out sex (string), out domicile (string), out
    postCode (long))

begin
        tmp_age: short          //stage (a)
        tmp_phoneNumber: long

        //Service 01; getUser(in id (long), out
            surName (string), out name (string),
            out age(short), out gender (string))
        invoke getUser( id, lastName, firstName,
            tmp_age, sex) //stage (b)
        age = (int)tmp_age   //stage (c)

        //Service02: obtainUserLocation(in id (
            long), out address (string), out
            postCode (long), out phoneNumber (long
            )))  )
        invoke obtainUserLocation( id, domicile,
            postCode, tmp_phoneNumber) //stage (b)
end
```

3) Development of a register to asset the structure and the set of items of each dimension. Specifically, a web form was defined where the experts could value the terms and synonyms included in the dictionary.

4) Delivery of the first version to the experts, including the semantic dictionary and the instructions to carry out the validation process.
5) Analysis of the evaluation data gathered from the experts.
6) Improvement of the semantic dictionary by means of including the modifications indicated by the group of experts.

In this sense, Runeson and Höst [21] indicate that in a case study context, a questionnaire (as mentioned at stage 3 of the procedure) could be used as a research instrument to acquire the relevant data from experts in the domain. We therefore drew up a questionnaire with which the experts could assess the suitability of each dictionary term and its synonyms. They could also suggest new terms for the dictionary and new synonyms for a particular dictionary term.

Since the domain of the case study was a university context, the experts recruited to respond to the questionnaire were university lecturers and researchers. The results were: (i) the suitability of each term was evaluated at an average rating of 3.65 out of 5, meaning that the terms used in the dictionary were appropriate; (ii) the suitability of each synonym with respect to the related term was evaluated at an average rating of 3.15 out of 5; and (iii) several new terms were proposed by the experts, many of which were included in the final dictionary.

It needs to be stressed that the dictionary is not only of importance for an adequate alignment process but that it is not written in stone. On the contrary, it is a living instrument that can be adapted in accordance to recommendations given by experts in the domain.

In sum, with respect to the research question RQ_1 – *Is the dictionary used during the alignment process valid with respect to the BP domain?* – the answer has to be that the dictionary used in the automatic alignment process is indeed valid, and that the dictionary terms and their synonyms are valid for describing the domain.

B. MANUAL ALIGNMENT BY EXPERTS

In the second step, a dossier with the relevant information about the BPs and the services available in the case study was prepared to be used by a group of experts versed in the subject matter. Concretely, this group of experts was composed of seventeen senior software engineers from different IT companies established in a Technological Park located in our Campus University. Specifically, the dossier that they received comprised the BPMN models of the case study and a list of the available underlying services.[3]

The main objective of this step was to measure the percentage of BPTs that were manually aligned with the underlying services. As mentioned above, the number of BPTs and services identified were 80 and 65 respectively. This alignment was carried out twice. The first time, the experts used the original description of the services. The second time, they had to use a random service signature, by which we mean that the original service signature was changed by replacing some of their parameters with synonyms, by altering the position of many parameters, and by replacing the data types of many parameters with other compatible types. Obviously, we expected that these changes would make the manual alignment process more difficult.

To carry out this validation process, the following procedure has been followed:

1) The procedure was explained to every expert.
2) The dossier with instructions and data about tasks and services was given to every expert.
3) Every expert completed the task. It should be noted that they complained about the excessive workload involved in the second execution. This second execution is the one with the randomized method signatures.
4) After gathering the alignment report from every expert, a comparative process was applied that met the next restriction: an alignment between a BPS and a service would only be right when, at least, 80% of experts had previously identified such alignment in their reports.

At an initial result, in the first execution, 75 out of 80 (93.75%) of the BPTs were aligned manually by the experts. In the second execution, with the randomized service signatures, the proportion dropped significantly to 80.00%. Also, the time the experts took to carry out the alignment was four times that taken in the first execution, reflecting the complication caused by the random signatures.

C. SEMIAUTOMATIC ALIGNMENT PROCESS

In the third and last step, the same BP models and underlying services used in Step 2 were aligned by our semiautomatic alignment process. Also as in Step 2, the alignment process was carried out twice: first, using the initial service descriptions, and secondly using random service signatures. We must note that an on-line dictionary different from that used in our alignment algorithm as used to obtain the synonym parameter names. This decision meant that the dictionaries used in the two processes (the alignment process and the process to change the service signatures) were not directly related.

Figure 8 shows the results for the incompatibilities detected and resolved. As one observes, various incompatibilities were identified and resolved, which would facilitate the alignment between the BPTs and the underlying services. Moreover, for the usual situation represented by the random service signatures, the results are the same as when the service signatures are more similar to the BPT signatures.

1) ANALYSIS OF THE RESULTS OF THE SEMIAUTOMATIC ALIGNMENT PROCESS

Next, we shall analyze the results of our semiautomatic alignment process, focusing on the unaligned tasks, false positives, and false negatives. This analysis will allow these results to be correlated with the experts' results.

[3]The dossier is available for review at `https://sites.google.com/site/migrasoa/uex-case-study/uex-validation`

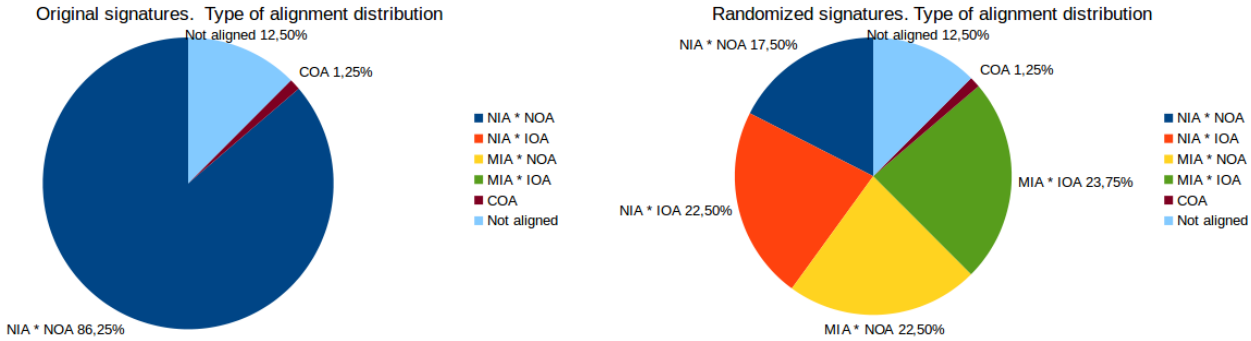

FIGURE 8. Summarized results of the alignment process carried out using our approach. Left: The alignment type distribution with the original service signatures. Right: The alignment type distribution with randomized service signatures.

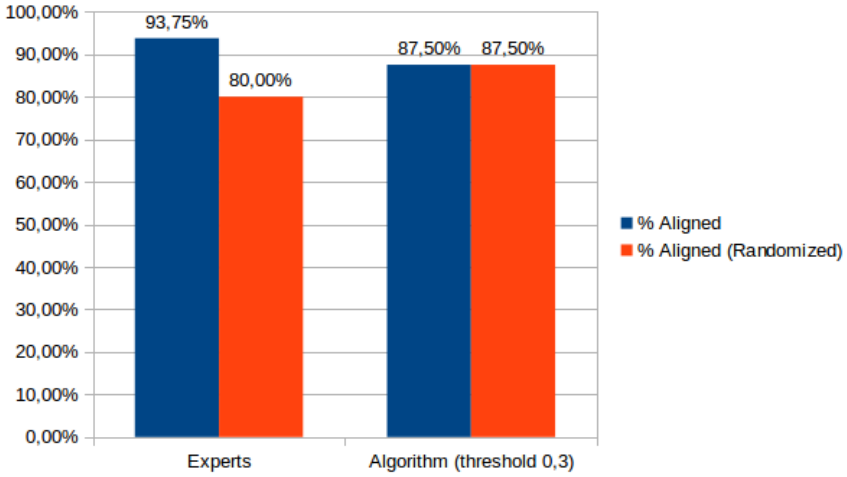

FIGURE 9. Comparison of the results of an expert manual alignment with those of the proposed algorithm.

The **unaligned tasks** mainly correspond to new tasks in the BPMN diagrams which could only be aligned with new service implementations or external services such as social network services (an example is the *Publish Facebook* task). In this sense, there is no difference between the unaligned BPTs of the experts and of our approach.

The results also include **false negatives**. A false negative is an unaligned task that has all its alignment values below the threshold, even though they apparently should have been aligned with some service. Examples of false negatives are the results of the *Exclude User* or *Exclude Library* tasks. The specific reason for these false negatives was that our dictionary did not include the *exclude* term (in the sense of the meaning of *remove* or *destroy*), but did include terms such as *delete* or *erase*. Thus, these examples indicate that, although the dictionary as a key element was validated by experts in the specific domain, it may still be incomplete. Other false negatives are mainly due to the alignment process not aligning terms successfully when the parameter names are compound or incomplete words (e.g., the parameter called *elem* instead of *element*). This is an open line of work to improve the proposal.

Finally, **false positives** are unusual due to the requirements of the alignment algorithm being so restrictive. A false positive occurs when there is an alignment between a task and a service that should not have happened, since their actions or objects do not have the same meaning or a similar meaning. For example, the alignment between the *Cancel Title* task and *deleteTitle* service produces a false positive. These terms are aligned because their actions are synonymous in the dictionary, even though their meaning is not suitable for our case study. Concretely, the *Cancel Title* task should be executed when we try to create a title that already exists. However, since *Cancel Title* task only requires to discard the new *Title*, it should not invoke the *deleteTitle* service because it would delete the current *Title*.

2) COMPARISON OF THE SEMIAUTOMATIC ALIGNMENT PROCESS'S RESULTS WITH THOSE OF THE EXPERTS

Next, we shall compare the alignment results obtained by our proposal with those of the experts. To this end, the principal measure we shall use is the percentage of alignment tasks. Figure 9 shows a comparative results between the BPTs manually aligned by the experts and by our proposal.

The results obtained in the first case (that with the original service description) show that the experts were able to align 93.75% of the BPTs, whereas the first execution of our

algorithm aligned 87.50%. The *false negatives* obtained were analyzed above.

The results obtained in the second case (that with the randomized service description) show that the percentage of the BPTs aligned using the algorithm, including the incompatibility identification and resolution, was still 87.50%. However, the experts' results decreased to a value of 80.00%. In this latter case, several BPTs were not aligned with the underlying service due to the difficulty of finding a suitable correspondence between the two. The experts described the alignment process with a random service signature as being like having to do a jigsaw puzzle.

With respect to RQ_2 – *Is the effort required to identify a potential alignment between a task and a service greater when the service parameter names, their order, or their data types are substantially different from those defined in the task?* – the answer has to be that it is indeed more costly to manually align BPTs and services when they have important, even though solvable, semantic differences. Specifically, the experts have the practical knowledge needed to align BPTs and the underlying services using semantic relations among terms or compatible types. However, complex but common situations such as random service signatures make it difficult for them to apply their practical knowledge, reducing their efficiency. Moreover, the experts must implement each separate adapter code (wrapper code) to resolve each incompatibility they identify.

With respect to RQ_3 – *Is the effort required to identify a potential alignment between a task and a service greater when they have different numbers of parameters?* – the results show that the effort required to manually align BPTs and services increases when: (i) the experts need to identify services in which several parameters are not required for the alignment, and (ii) it is necessary to develop a wrapper which facilitates the alignment between a task and a service. Specifically, the order or the number of the parameters are characteristics which the experts manage poorly, and consequently are error prone. In this sense, the results given by the alignment algorithm are stable in these same situations.

Finally, with respect to the main research question RQ_0 – *Can a semiautomatic alignment process between BPTs and the underlying services obtain results similar to those of the alignment carried out manually by an expert?* – the results obtained by the alignment algorithm are similar to those obtained by the experts. They may even be better when the complexity in identifying a correspondence between a BPT and an underlying service increases. An example of this situation was found in the second case of the study, when the random signatures of the services made their alignment difficult.

VI. CONCLUSIONS

The alignment of BPs with the underlying services or external services is a challenge for SOA architects. The architect faced tedious tasks as searching for, comparing, and selecting BPTs in order to align them with the available services. Also, several incompatibilities between BPTs and services need to be identified and resolved manually.

We have described a mathematical specification to describe formally how to identify and resolve several of these incompatibilities between BPTs and services. Then this mathematical specification was applied to design a semiautomatic process carrying out this identification and resolution, and improving the final alignment results. Note that the identification and resolution of incompatibilities are the key elements providing an advantage over the experts' results in the overall alignment process.

The research questions addressed in this work have helped shed some light on the relevance of a semiautomatic alignment process between BPs and the underlying services. The alignment process proposed in this paper is capable of giving results similar to those obtained by experts. The automation reduces the likelihood of error, improves productivity, and facilitates the alignment of large quantities of BPTs and services. Note that the alignment algorithm's results are stable in complex situations in which the definition of the BPTs and the service signatures are in principle incompatible in terms of semantic issues, parameter order, compatible parameter types, or extra parameters.

Future work will include new case studies and a proof-of-concept study about how method signatures or parameter names are defined in large systems, for example, in some open-source code project. This will allow the recognition of methods or parameters to be improved, which consequently would also improve the alignment process itself. Other open research will focus on the identification of services that are directly available from the services repository under such common protocols as SOAP or REST. This would make more underlying services available for alignment. Moreover, although no semantic Web services are usually available, we shall explore their use as a basis for the service layer because this should improve the results of the alignment process. Finally, the proposed alignment process should be used to instantiate automatically a BP Management System (BPMS) such as Activiti or Intalio, facilitating BP deployment and monitoring.

REFERENCES

[1] A. Ullah and R. Lai, "A systematic review of business and information technology alignment," *ACM Trans. Manage. Inf. Syst.*, vol. 4, no. 1, p. 4, 2013.

[2] J. Zdravkovic, M. Henkel, and P. Johannesson, "Moving from business to technology with service-based processes," *IEEE Internet Comput.*, vol. 9, no. 3, pp. 73–81, May 2005.

[3] P. Harmon, *Business Process Change: A Guide for Business Managers and BPM and Six Sigma Professionals*. San Mateo, CA, USA: Morgan Kaufmann, 2010.

[4] K. A. Alam, R. Ahmad, A. Akhunzada, M. H. N. M. Nasir, and S. U. Khan, "Impact analysis and change propagation in service-oriented enterprises: A systematic review," *Inf. Syst.*, vol. 54, pp. 43–73, Dec. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437915001179

[5] J. Luftman and T. Brier, "Achieving and sustaining business-IT alignment," *California Manage. Rev.*, vol. 42, no. 1, pp. 109–122, 1999.

[6] B. H. Reich and I. Benbasat, "Factors that influence the social dimension of alignment between business and information technology objectives," *MIS Quart.*, vol. 24, no. 1, pp. 81–113, 2000.

[7] Z. Yan, R. Dijkman, and P. Grefen, "Business process model repositories—Framework and survey," *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 380–395, 2012.

[8] J. Dorn, C. Grün, H. Werthner, and M. Zapletal, "From business to software: A B2B survey," *Inf. Syst. e-Bus. Manage.*, vol. 7, no. 2, pp. 123–142, 2009.

[9] A. Ullah and R. Lai, "A requirements engineering approach to improving IT-business alignment," in *Information Systems Development*. New York, NY, USA: Springer, 2011, pp. 771–779.

[10] (Jan. 2011). *BPMN2.0*. [Online]. Available: http://www.omg.org/spec/bpmn/2.0/

[11] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-oriented Architecture Best Practices*. Upper Saddle River, NJ, USA: Prentice-Hall, 2005.

[12] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. WISE*, Dec. 2003, pp. 3–12.

[13] N. Joachim, "A literature review of research on service-oriented architectures (SOA): Characteristics, adoption determinants, governance mechanisms, and business impact," in *Proc. 17th Amer. Conf. Inf. Syst. (AMCIS)*, Detroit, MI, USA, V. Sambamurthy and M. Tanniru, Eds. Atlanta, GA, USA: Association for Information Systems, 2011.

[14] A. Becker, P. Buxmann, and T. Widjaja, "Value potential and challenges of service-oriented architectures—A user and vendor perspective," in *Proc. 17th Eur. Conf. Inf. Syst. (ECIS)*, Verona, Italy, S. Newell, E. A. Whitley, N. Pouloudi, J. Wareham, and L. Mathiassen, Eds. Atlanta, GA, USA: AIS, 2009, pp. 2085–2096. [Online]. Available: http://is2.lse.ac.uk/asp/aspecis/20090173.pdf

[15] P. Vitharana, K. Bhaskaran, H. K. Jain, H. J. Wang, and J. L. Zhao, "Service-oriented enterprises and architectures: State of the art and research opportunities," in *Proc. 13th Amer. Conf. Inf. Syst. (AMCIS)*, Keystone, CO, USA, J. A. Hoxmeier and S. Hayne, Eds. Atlanta, GA, USA: Association for Information Systems, 2007, p. 343. [Online]. Available: http://aisel.aisnet.org/amcis2007/

[16] P. A. Laplante, J. Zhang, and J. Voas, "What's in a name? Distinguishing between SaaS and SOA," *IT Prof.*, vol. 10, no. 3, pp. 46–50, May 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MITP.2008.60

[17] S. Fan, J. Zhao, W. Dou, and M. Liu, "A framework for transformation from conceptual to logical workflow models," *Decis. Support Syst.*, vol. 54, no. 1, pp. 781–794, 2012.

[18] J. Davis, *Open Source SOA*, 1st ed. Greenwich, CT, USA: Manning, 2009.

[19] M. Weske, *Business Process Management—Concepts, Languages, Architectures*, 2nd ed. Berlin, Germany: Springer, 2012.

[20] E. Sosa, P. J. Clemente, M. Sánchez-Cabrera, J. M. Conejero, R. Rodríguez-Echeverría, and F. Sanchez-Figueroa, "Service discovery using a semantic algorithm in a SOA modernization process from legacy Web applications," in *Proc. IEEE 10th World Congr. Services (SERVICES)*, Jun. 2014, pp. 470–477.

[21] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.

[22] A. Perin-Souza and R. J. Rabelo, "A model for dynamic services discovery over largely distributed providers based on QoS and business processes contexts," in *Proc. SERVICES*, Jul. 2011, pp. 347–354.

[23] T. Narock, V. Yoon, and S. March, "A provenance-based approach to semantic Web service description and discovery," *Decis. Support Syst.*, vol. 64, pp. 90–99, Aug. 2014.

[24] K. Belhajjame *et al.*, "PROV-O: The PROV Ontology," World Wide Web Consortium, Cambridge, MA, USA, Tech. Rep. prov-o, 2012. [Online]. Available: http://www.w3.org/TR/prov-o/

[25] S. Zednik, P. Fox, D. L. McGuinness, P. P. da Silva, and C. Chang, "Semantic provenance for science data products: Application to image data processing," in *Proc. 1st Int. Workshop Role Semantic Web Provenance Manage. (SWPM) 8th Int. Semantic Web Conf. (ISWC)*, Washington, DC, USA, vol. 526, J. Freire, P. Missier, and S. S. Sahoo, Eds. Aachen, Germany: CEUR-WS.org, Oct. 2009, pp. 4–10.

[26] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for Web services," in *Proc. 13th Int. Conf. Very Large Data Bases (VLDB)*, vol. 30, 2004, pp. 372–383.

[27] O. Hatzi, G. Batistatos, M. Nikolaidou, and D. Anagnostopoulos, "A specialized search engine for Web service discovery," in *Proc. IEEE 19th Int. Conf. Web Services*, Honolulu, HI, USA, Jun. 2012, pp. 448–455.

[28] M. Klusch, B. Fries, and K. P. Sycara, "Automated semantic Web service discovery with OWLS-MX," in *Proc. 5th Int. Joint Conf. Auto. Agents Multiagent Syst. (AAMAS)*, Hakodate, Japan, May 2006, pp. 915–922.

[29] G. C. Hobold and F. Siqueira, "Discovery of semantic Web services compositions based on SAWSDL annotations," in *Proc. IEEE 19th Int. Conf. Web Services*, Honolulu, HI, USA, Jun. 2012, pp. 280–287.

[30] C. Li, R. Zhang, J. Huai, X. Guo, and H. Sun, "A probabilistic approach for Web service discovery," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2013, pp. 49–56.

[31] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin, "Semantic Web service discovery using natural language processing techniques," *Expert Syst. Appl.*, vol. 40, no. 11, pp. 4660–4671, 2013.

[32] Princeton University. (2014). *WordNet Project*. [Online]. Available: http://wordnet.princeton.edu

[33] O. El-Gayar and A. Deokar, "A semantic service-oriented architecture for distributed model management systems," *Decis. Support Syst.*, vol. 55, no. 1, pp. 374–384, 2013.

[34] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella, "When are two Web services compatible?" in *Technologies for E-Services* (Lecture Notes in Computer Science), vol. 3324, M.-C. Shan, U. Dayal, and M. Hsu, Eds. Berlin, Germany: Springer, 2005, pp. 15–28.

[35] M. Dumas, B. Benatallah, and H. R. M. Nezhad, "Web service protocols: Compatibility and adaptation," *IEEE Data Eng. Bull.*, vol. 31, no. 3, pp. 40–44, Sep. 2008.

[36] A. Ait-Bachir and M.-C. Fauvet. (2009). *Diagnosing and Measuring Incompatibilities Between Pairs of Services*. [Online]. Available: http://hal.inria.fr/docs/00/95/38/47/PDF/dexa09-ait-bachir.pdf

[37] X. Li, Y. Fan, S. E. Madnick, and Q. Z. Sheng, "A pattern-based approach to protocol mediation for Web services composition," *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 304–323, 2010, doi: 10.1016/j.infsof.2009.11.002.

[38] Y. Taher, A. Ait-Bachir, M.-C. Fauvet, and D. Benslimane, "Diagnosing incompatibilities in Web service interactions for automatic generation of adapters," in *Proc. IEEE 23rd Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Bradford, U.K., May 2009, pp. 652–659. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/AINA.2009.118

[39] S. R. Ponnekanti and A. Fox, "Interoperability among independently evolving Web services," in *Middleware 2004* (Lecture Notes in Computer Science), vol. 3231. Berlin, Germany: Springer, 2004, pp. 331–341.

[40] H. Kaindl, R. Hoch, and R. Popp, "Semantic task specification in business process context," in *Proc. 11th Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, May 2017, pp. 286–291.

[41] E. Sosa, P. J. Clemente, J. M. Conejero, R. Rodríguez-Echeverría, "A model-driven process to modernize legacy Web applications based on service oriented architectures," in *Proc. 15th IEEE Int. Symp. Web Syst. Evol. (WSE)*, Eindhoven, The Netherlands, Sep. 2013, pp. 61–70.

[42] T. Baier, C. Di Ciccio, J. Mendling, and M. Weske, "Matching events and activities by integrating behavioral aspects and label analysis," *Softw. Syst. Model.*, vol. 17, no. 2, pp. 573–598, 2018.

[43] J. Z. Wang, F. Ali, and R. Appaneravanda, "A Web service for efficient ontology comparison," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2005, pp. 843–844.

[44] A. Tversky, "Features of similarity," *Psychol. Rev.*, vol. 84, no. 4, p. 327, 1977.

[45] J. Cardoso, "Discovering semantic Web services with and without a common ontology commitment," in *Proc. IEEE Services Comput. Workshops (SCW)*, Sep. 2006, pp. 183–190.

[46] S. Cubo, B. Martín, and J. L. Ramos, *Métodos de Investigación y Análisis de Datos en Ciencias Sociales y de la Salud*, 1st ed. Madrid, Spain: Ediciones Pirámide, 2011.

ENCARNA SOSA-SÁNCHEZ received the B.Sc. and Ph.D. degrees in computer science from the University of Granada, in 1995 and 2018, respectively. She is currently pursuing the Ph.D. degree with the Computer Science Department, University of Extremadura, Spain. She is also an Assistant Professor with the Computer Science Department, University of Extremadura. She has published several peer-reviewed papers in international journals, workshops, and conferences. She is involved in several research projects. Her research interests include service-oriented architectures, business process modeling, and model-driven development.

PEDRO J. CLEMENTE received the B.Sc. and Ph.D. degrees in computer science from the University of Extremadura, in 1998 and 2007, respectively. He is currently an Associate Professor with the Computer Science Department, University of Extremadura, Spain. He has published numerous peer-reviewed papers in international journals, workshops, and conferences. He is involved in several research projects. His research interests include component-based software development, aspect orientation, service-oriented architectures, business process modeling, and model-driven development. He has participated in many workshops and conferences as a speaker and is a member of the program committees.

ÁLVARO E. PRIETO received the B.Sc. and Ph.D. degrees in computer science from the University of Extremadura, in 2000 and 2013, respectively. He is currently an Assistant Professor of computer languages and systems with the University of Extremadura, Spain. He is also a member of the Quercus Software Engineering Group. He is involved in various research, development, and innovation projects. His research interests include ontologies, workflows, and linked open data.

CARMEN ORTIZ-CARABALLO received the Ph.D. degree in mathematics from the University of Seville, in 2011. She is currently an Assistant Professor of mathematics with the University of Extremadura, Spain. She has published several peer-reviewed papers in international journals, workshops, and conferences on harmonic analysis. She is involved in several research projects. Her research interests include harmonic analysis and applied mathematics, in which she is currently collaborating with different research groups.